1988

Design and implementation of a prototype graphical user interface for a model management system

Wyant, Marvin Abram

Monterey, California : Naval Postgraduate School

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School 6b OFFICE SYMBOL (If applicable): Code 54	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO | PROJECT NO | TASK NO | WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
DESIGN AND IMPLEMENTATION OF A PROTOTYPE GRAPHICAL USER INTERFACE FOR A MODEL MANAGEMENT SYSTEM

12. PERSONAL AUTHOR(S)
Wyant, Marvin Abram Jr.

13a. TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988 March	106

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Models; Structured models; Model management system; MMS; User interface; Prototype; Graphics; System design; DBMS; Database

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this thesis is to design and implement a prototype graphical user interface for a structured model management system. The program is written for an IBM PC using Lattice-C, the Halo graphics package, and the ORACLE DBMS. Design and Implementation issue are discussed and evaluated. Future enhancements to the program and a recommendation as to the disposition of the prototype are also included.

A brief explanation of structured modeling is presented. An example problem is used to illustrate the various model representation provided by structured modeling. The progra re-creates the graphical representations of structured modeling from a database representation.

The results of this thesis show that the prototype design methodology is an excellent supplement to the traditional life-cycle design methodology. The implications of this observation are discussed in relationship to the graphical user interface program.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: Prof. Daniel R. Dolk	22b TELEPHONE (Include Area Code): (408) 646-2260 | 22c OFFICE SYMBOL: Code 54Dk

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

Design and Implementation of a Prototype Graphical User Interface for a Model Management System

by

Marvin Abram Wyant, Jr.
Lieutenant, United States Navy
B.S., Texas A&M University, 1982

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1988

ABSTRACT

The purpose of this thesis is to design and implement a prototype graphical user interface for a structured model management system. The program is written for an IBM PC using Lattice-C, the Halo graphics package, and the ORACLE DBMS. Design and implementation issues are discussed and evaluated. Future enhancements to the program and a recommendation as to the disposition of the prototype are also included.

A brief explanation of structured modeling is presented. An example problem is used to illustrate the various model representations provided by structured modeling. The program re-creates the graphical representations of structured modeling from a database representation.

The results of this thesis show that the prototype design methodology is an excellent supplement to the tradition life-cycle design methodology. The implications of this observation are discussed in relationship to the prototype graphical user interface program.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

Model management systems (MMS) can be effective tools within an organization's information system. Top-level management can use "what if" type models and corporate data for strategic decision making. At lower levels, modeling may aid in the optimization of an assembly process using production data as inputs.

Management's awareness of the need for MMS has increased due to several factors currently facing the management science and operations research (MS/OR) communities [Ref. 1: pp. 547-549]:

- MS/OR activities are not highly productive.
- Managerial acceptance of model-based assistance is low.
- The micro-computer revolution provides the medium for greater productivity and acceptance of MS/OR applications.
- The advances in database management systems (DBMS) technology provide a suitable interface for the high data demands of MS/OR software.
- The popularity of spreadsheet modeling proves that many people have the potential to construct useful models given the appropriate tool.

Professor A. M. Geoffrion of UCLA proposes structured modeling as a theoretical base from which these issues can be addressed [Ref. 1].

The main purpose of this thesis is to design and implement a prototype of a graphical user interface for an MMS based on the principles of structured modeling. Of interest also will be the refined requirements specifications based on the implementation of the prototype system.

B. STRUCTURED MODELING

Structured modeling provides the theoretical foundation for a new generation of MMS. Its goal is to increase the productivity and acceptance of MS/OR applications through greater usage and understanding by model practitioners and non-practitioners alike. Structured modeling allows for the inclusion of several desirable features necessary for an effective MMS [Ref. 1: p. 550].

The conceptual framework of structured modeling provides for the delineation of a wide range of MS/OR mathematical models in a single representation format. The representation of a structured model is independent of both its solution operators and its required data. The representation format lends itself to computer implementation with a graphical user interface. The independence of a model and its data allows the integration with current database technology. Finally, structured modeling provides support for the complete life-cycle of a model [Ref. 2: p. 1-2].

C. PROGRAMMING ENVIRONMENT

The hardware configuration for the implementation of this system is an IBM-PC, or compatible, equipped with an enhanced graphics adapter (EGA) card. The selection of this configuration, vice a mainframe configuration, is due to the widespread use of micro-computers throughout many organizations. The higher resolution graphics attainable from the EGA card improves the aesthetic quality of the interface. These requirements are not excessively restrictive and are comparable with limitations imposed by commercial software products.

The programming environment for the system is the Lattice-C compiler along with the HALO graphics package and the PC ORACLE DBMS. The Lattice-C compiler gives programming versatility not found in many micro-computer language implementations. The HALO graphics package offers a complete library of high speed graphics routines designed for the Lattice-C compiler. The selection of the PC ORACLE DBMS is two-fold: (1) its support of SQL (Standard Query Language) and (2) compatibility with the Lattice-C compiler.

D. SUMMARY

The purpose of this thesis is to implement a prototype graphical user interface for an MMS based upon the principles of structured modeling. The prototype will be used to evaluate the initial requirements specifications for the interface. Errors and problems encountered in implementation

of the prototype will be reported. Refined system requirements will also be presented. Recommendations for further research and implementation of the interface will be discussed. Specific emphasis will be placed on whether to continue the system implementation from the prototype or begin anew using conventional structured analysis and design techniques.

Chapter II provides an overview of structured modeling. The basic descriptions of a structured model's elements, representations, and underlying principles are discussed. An example of a structured model is constructed to reinforce the theoretical concepts.

Chapter III presents the initial system requirements and proposed design of the graphical user interface prototype. The data structures and critical modules are detailed in this chapter.

Chapter IV discusses problems that were encountered during the implementation of the prototype. A discussion of refinements to the initial system requirements, reusability of the prototype, and integration with other concurrent efforts is included.

Chapter V concludes this thesis with a synopsis of the results, observations and recommendations for future study.

II. STRUCTURED MODELING

A. INTRODUCTION

Two factors hinder the proliferation of management science and operations research (MS/OR) modeling: the low productivity of MS/OR activities and a lack of managerial acceptance of modeling [Ref. 1: pp. 547-549]. The factors interrelate in that low productivity amplifies management's reluctance to accept models.

Low MS/OR productivity is inherent in the current modeling technology. Present modeling practices that adversely affect MS/OR productivity are:

- The redundancy of effort needed to place models in the required representation formats. Typically, external requirements force the generation of three separate model representations. Communication with non-MS/OR personnel mandates a logical model representation. A mathematical representation is required for analysis. Finally, computational complexity and data requirements necessitate a representation that is computer-executable.
- The difficulty of interfacing logical and mathematical model representations with available modeling software. Most MS/OR software products use no interface standards, accept only one type of model schema, and do not support the entire modeling life-cycle.

One might expect that the lack of managerial acceptance of models is due to models not capturing reality adequately, however, this is not generally the case. The issue of acceptance is largely a matter of management's reluctance in becoming dependent on MS/OR personnel. Modeling

practitioners do not convey the structure of models in formats that are easy to comprehend. Therefore, the operation of the model is not understandable and results must be blindly accepted. This situation is not conducive to proper management.

The quest for better MS/OR productivity and acceptance, however, is far from hopeless. Several recent technological advances present opportunities for improvement in modeling techniques and comprehension [Ref. 1: pp. 548-549]. The increased availability and usage of micro-computers provides an instrument to raise productivity. Accessing data through database management systems allows use of a single data model to serve many applications. Finally, the overwhelming popularity of spreadsheet software suggests greater acceptance and demand for models when represented in an understandable format.

A new generation of modeling systems must be developed that resolve MS/OR and management conflicts and incorporate the new technologies. To effectively address these issues, the new systems should possess the following features [Ref. 1: p. 549]:

1. A single model representation supporting logical views, mathematical analysis, and computer execution.
2. A model representation sufficiently general to encompass a majority of MS/OR models.
3. An interface that supports the entire modeling lifecycle.

4. Adaptability to a micro-computer version with a modern user interface.

5. Integration with a database management system.

6. Instantaneous solutions in the tradition of spreadsheet software.

Structured modeling lays the foundation for a modeling system providing all of these features.

This chapter gives a basic description of structured modeling. The thrust is to define terms and present the underlying concepts of structured modeling. A simple transportation model will be used to reinforce and supplement the explanation of concepts. The intention of this chapter is not to cover structured modeling in its entirety. Readers requiring a comprehensive treatment of structured modeling should consult the research of Geoffrion [Ref. 1: pp. 552-563], [Ref. 2: pp. 2-1 to 2-101].

B. PRINCIPLES OF STRUCTURED MODELING

> Structured modeling is a unified modeling framework based on acyclic, attributed graphs to represent cross-references between elements of a model, and hierarchies to represent levels of abstraction. [Ref. 3: p. 4]

A structured model consists of three basic structures: an elemental structure, a generic structure, and a modular structure.

A structured model is expressed in terms of five types of elements: primitive entities, compound entities, attributes, functions, and tests. Every structured model contains at least one primitive entity. A primitive entity defines the

existence of an object. Its description makes no reference to a value or magnitude. Compound entities refer to other entities, usually primitive entities, to define a new and unambiguous entity. Attributes associate values and properties with entity type elements. Variable attributes are extensions of attribute elements. The values of a variable attribute are likely to change and come under the control of a model solver. The value determination of a function element is in terms of a rule or equation. A test element is essentially a function element returning a boolean or true/false value.

A model's elemental structure is defined as a non-empty, finite, closed, acyclic collection of model elements [Ref. 3: p. 4]. Every element within an elemental structure, except primitive entities, has a calling sequence. An element calls another element if the second (or called) element is in the calling sequence of the first element. Calling sequences provide a means of capturing the cross-references between the elements of a model. An acyclic elemental structure will have no element which directly or indirectly calls itself. Acyclicity prevents a model from being indeterminable [Ref. 2: p. 2-4]. The graphical representation of the elemental structure contains all the model's elements and accurately depicts each elements calling sequence.

The generic structure of a model is an abstraction of the elemental structure. Similar element types are grouped into

genera (or partitions) based on generic similarity. Each element belongs to only one genus (singular of genera). Satisfaction of generic similarity occurs if all members of a genus are of the same element type, have an equal number of calling sequence segments, and make calls to the same genus. The graphical representation of the generic structure, the genus graph, shows each genus as a node and the relationships between the genera as directed arcs between the appropriate nodes.

The modular structure provides a tree-type representation of a model which serves as an aggregation of the generic structure. All terminal nodes in the structure are genera and all non-terminal nodes are modules. Modules are meaningful groupings of genera. Modular structures satisfy the conditions of monotone ordering. Monotone ordering requires that modular structures fit an indented list format with no genera making forward references to any other genera. This implies that a module is essentially a hierarchically ordered list of genera.

C. EXAMPLE: THE TRANSPORTATION MODEL

The best way to gain an understanding of structured modeling is with an example. This section uses the transportation model to reinforce the definitions and concepts of structured modeling. The analysis of the transportation model begins with identification of the elements in the model. The elemental, generic, and modular structures will be derived

and illustrated. In addition, reachability and adjacency matrices are introduced as adjuncts to structured modeling. An alternative to the graphical representation of structured models is also presented.

The transportation model used for this example was first used by Geoffrion [Ref. 2: pp. 2-70 to 2-71]. The problem statement is as follows:

- Two production plants, one in Dallas and the other in Chicago, must supply goods to customers in Pittsburgh, Atlanta, and Cleveland.
- The production capacities of the Dallas and Chicago plants are 20,000 and 42,000 units, respectively.
- The number of units required by the customers in Pittsburgh, Atlanta, and Cleveland are 25,000, 15,000, and 22,000, respectively.
- The Dallas plant may supply each of the three cities, whereas, the Chicago plant may only supply the customers in Pittsburgh and Cleveland.
- The cost of shipping from Dallas to Pittsburgh is $23.50/unit; Dallas to Atlanta, $17.95/unit; Dallas to Cleveland, $32.45/unit; Chicago to Pittsburgh, $7.60/unit; Chicago to Cleveland, $25.75/unit.

The purpose is to determine a solution that provides all the customers with the necessary amount of goods at the lowest possible total cost.

The easiest elements in a model to define are primitive entities. The following list shows that there are five primitive entities in the transportation model:

- There exists a production plant in Dallas (DAL).
- There exists a production plant in Chicago (CHI).
- There exists a customer in Pittsburgh (PITT).

- There exists a customer in Atlanta (ATL).
- There exists a customer in Cleveland (CLE).

Notice there are no values associated with the primitive entities. Their existence is all that is necessary. The abbreviations in parentheses will be used to shorten future references in the definitions of new elements.

There are five other elements in the problem that have no value and are existential in nature. However, since these elements reference primitive entities, they must be of the compound entity type. The definition of compound entities is as follows:

- There exists a shipping link from DAL to PITT (LINK1).
- There exists a shipping link from DAL to ATL (LINK2).
- There exists a shipping link from DAL to CLE (LINK3).
- There exists a shipping link from CHI to PITT (LINK4).
- There exists a shipping link from CHI to CLE (LINK5).

Attribute elements associate values with entity type elements. There are numerous attribute elements in the transportation model:

- The production capacity of DAL is 20,000 units (SUPPLY1).
- The production capacity of CHI is 42,000 units (SUPPLY2).
- PITT requires 25,000 units (DEMAND1).
- ATL requires 15,000 units (DEMAND2).
- CLE requires 22,000 units (DEMAND3).
- The cost of using LINK1 is $23.50/unit (RATE1).
- The cost of using LINK2 is $17.75/unit (RATE2).

- The cost of using LINK3 is $32.45/unit (RATE3).
- The cost of using LINK4 is $7.60/unit (RATE4).
- The cost of using LINK5 is $25.75/unit (RATE5).

The transportation model also has a set of variable attributes. Each shipping link has an associated amount of flow (in number of units). The model solver determines the flow during model execution, therefore, flow is a variable attribute. The listing of variable attributes for shipping flow are:

- There is a shipping flow over LINK1 (FLOW1).
- There is a shipping flow over LINK2 (FLOW2).
- There is a shipping flow over LINK3 (FLOW3).
- There is a shipping flow over LINK4 (FLOW4).
- There is a shipping flow over LINK5 (FLOW5).

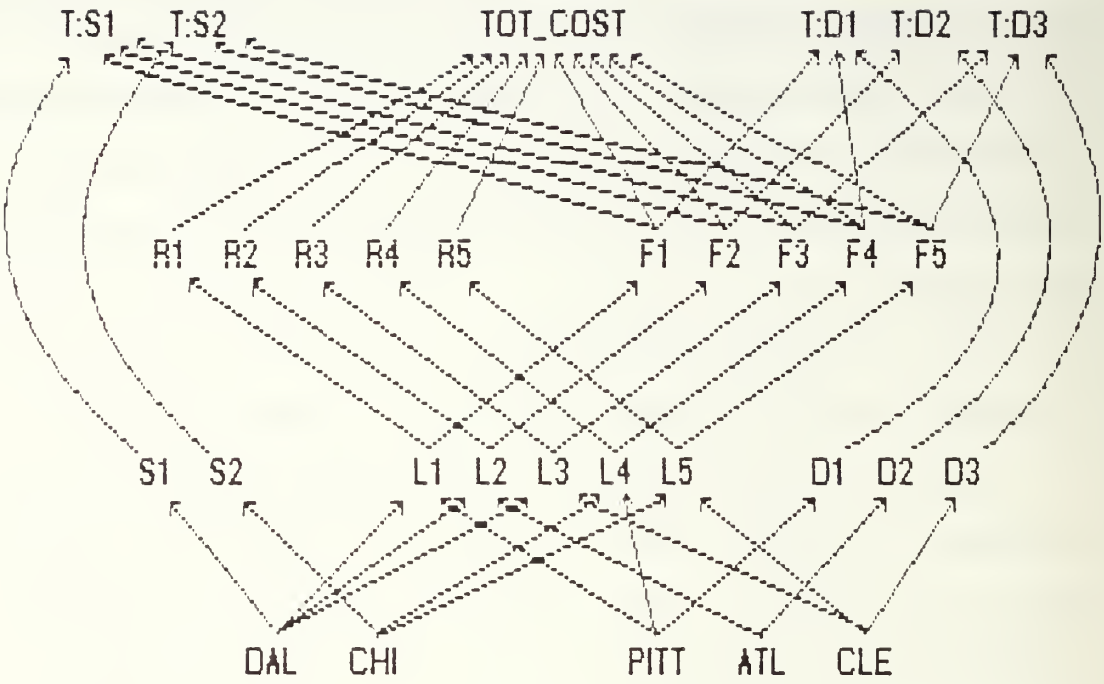
A function element exists in the transportation model which is the total cost of using each of the shipping links. Total cost is necessarily a function element due to its computational nature. There are also test elements to ensure that flows remain within the limits imposed by plant capacities and customer demands. Expressed in terms of the problem, the test and function elements are:

- There is a total cost (TOT_COST) which is the sum of the products of the individual flows and corresponding rates.
- The flow leaving DAL cannot exceed SUPPLY1 (T:SUPPLY1).
- The flow leaving CHI cannot exceed SUPPLY2 (T:SUPPLY2).
- The flow arriving at PITT must equal DEMAND1 (T:DEMAND1).
- The flow arriving at ATL must equal DEMAND2 (T:DEMAND2).

- The flow arriving at CLE must equal DEMAND3 (T:DEMAND3).

The construction of the elemental structure involves transcribing the calling sequences for each element into a graphical format. The graphical representation of an element may be a word, abbreviation, shape, or anything that is uniquely identifiable. The representation of the calling sequences are directed line segments. The base of the line segment identifies the called element and the head points to the calling element. Figure 2.1 shows the elemental structure for the transportation model. The depiction of relationships between elements are complete and accurate.

A generic model structure is a generalization of the elemental structure. The elements in a model are grouped by generic similarity. For example, a genus called FLOW is formed by grouping all the variable attribute elements describing the flow of goods across shipping links. Each of these elements have the same number of calling sequence segments to the same genus of elements. Thus, the grouping of FLOW1, FLOW2, FLOW3, FLOW4, and FLOW5 satisfies generic similarity requirements and forms a genus. Figure 2.2 shows the groupings of all the transportation model elements into their respective genera. The generic structure for the transportation model is shown as the genus graph in Figure 2.3. The T:GENUS_NAME is a convention for naming test genera.



KEY

S - SUPPLY
D - DEMAND
L - LINK
R - RATE
F - FLOW
T:S - T:SUPPLY
T:D - T:DEMAND

Figure 2.1 Elemental Structure: Transportation Model

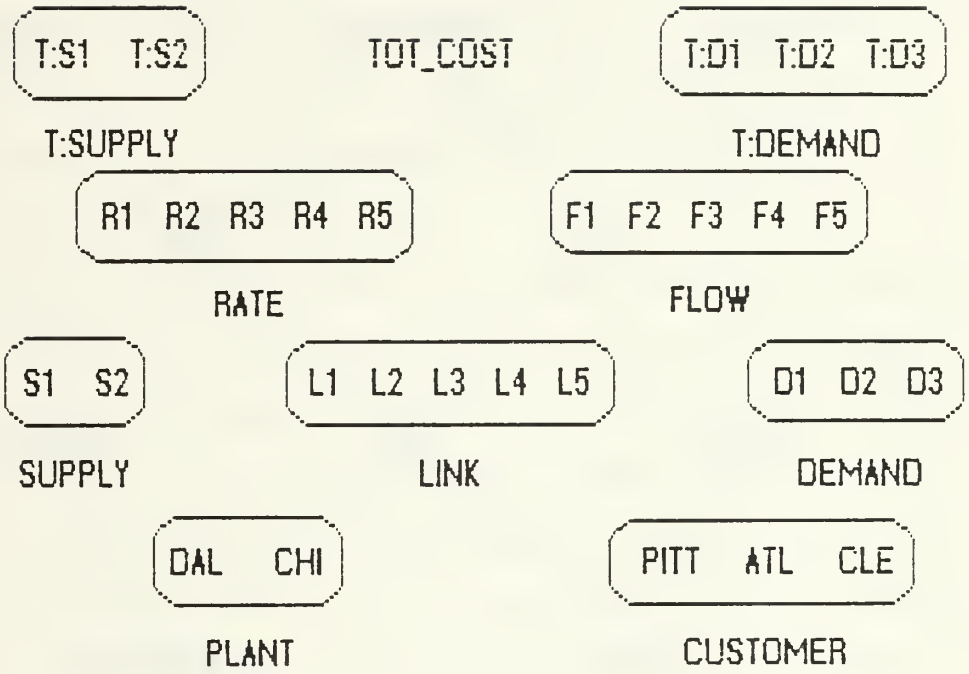


Figure 2.2 Genus Groupings: Transportation Model

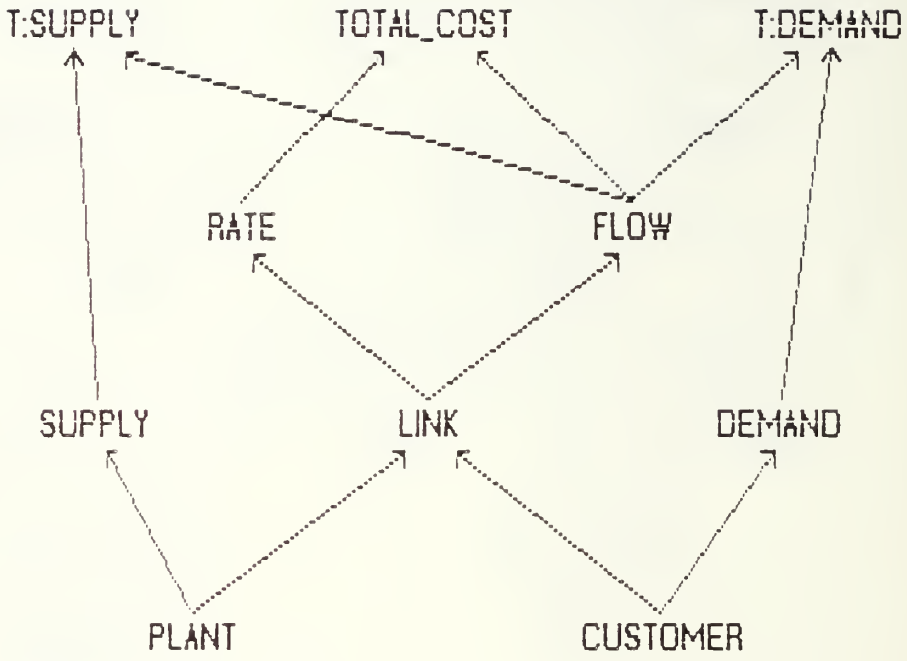


Figure 2.3 Genus Graph: Transportation Model

Modules are the basis of module structures. A module is a meaningful grouping of genera. The definition of a meaningful grouping is a function of the use of the modular structure. The only restriction on a modular structure is that it satisfies the requirement of monotone ordering. Figure 2.4 presents two acceptable representations of a modular structure, a tree structure and a modular outline. The ampersand preceding each module name is a structured modeling labeling convention. Note that all non-terminal nodes are modules, all terminal nodes are genera, and the monotone ordering is preserved.

Structured modeling offers other miscellaneous constructs that are useful in the formulation and analysis of models. A view of a model is simply a generic structure with a module replacing some of its corresponding genera. This construct can be useful in effective communication of models to non-MS/OR personnel by suppressing unnecessary details. Reachability and adjacency matrices provide a quick reference for determining the inter-relationships between model elements. Figures 2.5 and 2.6 show the adjacency and reachability matrices for the transportation model's generic structure. The convention for reading the matrices is that the column is the calling element and the row is the called element. A "1" in an adjacency matrix means that the element in that row is in the calling sequence of the column element. In a

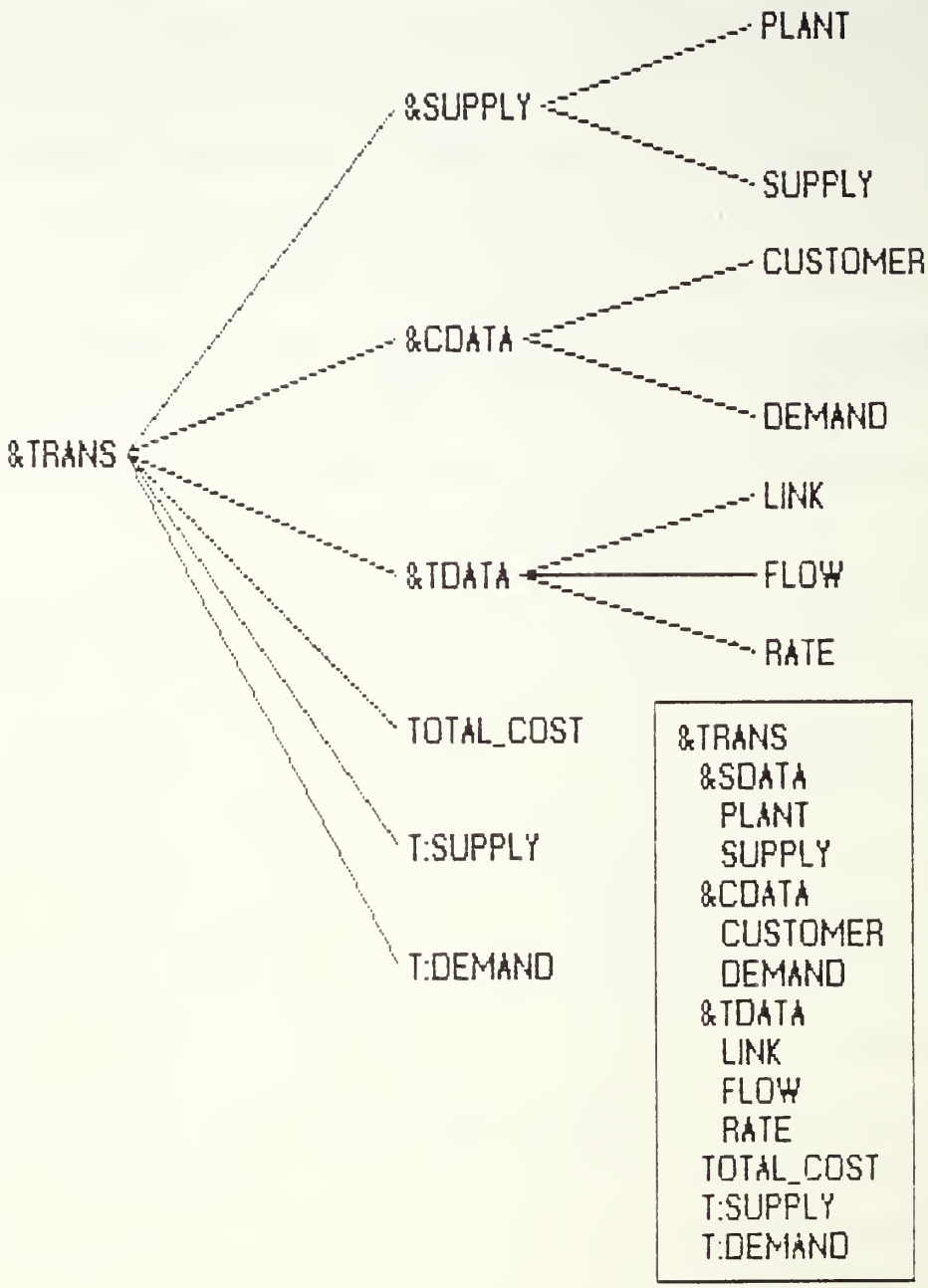


Figure 2.4 Modular Structure and Modular Outline: Transportation Model

	PLANT	SUPPLY	CUSTOMER	DEMAND	LINK	FLOW	RATE	TOTAL COST	T:SUPPLY	T:DEMAND
PLANT	.	1	.	.	1	.	.	.	.	.
SUPPLY	.	.	.	.	.	.	.	.	1	.
CUSTOMER	.	.	.	1	1	.	.	.	.	.
DEMAND	.	.	.	.	.	.	.	.	.	1
LINK	.	.	.	.	.	1	1	.	.	.
FLOW	.	.	.	.	.	.	.	1	1	1
RATE	.	.	.	.	.	.	.	1	.	.
TOTAL COST	.	.	.	.	.	.	.	.	.	.
T:SUPPLY	.	.	.	.	.	.	.	.	.	.
T:DEMAND	.	.	.	.	.	.	.	.	.	.

Figure 2.5 Adjacency Matrix: Transportation Model

	PLANT	SUPPLY	CUSTOMER	DEMAND	LINK	FLOW	RATE	TOTAL COST	T:SUPPLY	T:DEMAND
PLANT	1	1	.	.	1	1	1	1	1	1
SUPPLY	.	1	.	.	.	.	.	.	1	.
CUSTOMER	.	.	1	1	1	1	1	1	1	1
DEMAND	.	.	.	1	.	.	.	.	.	1
LINK	.	.	.	.	1	1	1	1	1	1
FLOW	.	.	.	.	.	1	.	1	1	1
RATE	.	.	.	.	.	.	1	1	.	.
TOTAL COST	.	.	.	.	.	.	.	1	.	.
T:SUPPLY	.	.	.	.	.	.	.	.	1	.
T:DEMAND	.	.	.	.	.	.	.	.	.	1

Figure 2.6 Reachability Matrix: Transportation Model

reachability matrix, a "1" means that the row and column elements are related, with the column element being higher in the hierarchy. This alternative model conceptualization is a familiar software engineering analysis tool.

Module and genus paragraphs supplement the graphical representation of structured models. The paragraphs provide more detailed interpretations than permitted graphically. Paragraphs are arranged and indented identically to modular outlines. A model schema is a complete collection of paragraphs for an entire model.

Structured modeling provides a highly developed syntax for genus and module paragraphs. The paragraph notation presents the model information in a compact form suitable for computer parsing. In Figure 2.7, the generalized paragraph notations are shown. Figure 2.8 is the schema for the transportation model. The following comments explain the notational conventions:

- The module and genus names appear exactly as they do in the module structure. Both module and genus names should be short, descriptive, and upper case.
- The "i" indicates that there is a symbolic genus index. The symbolic genus index is a referencing system to identify elements within a genus. For example, if DAL is $PLANT_1$ and CLE is $CUST_3$, then, $LINK_{13}$ refers to the transportation link between DAL and CLE.
- The genus type is identified by an appropriate abbreviation within a set of slashes.
- The index set statement defines the population of elements associated with the genus.

PARAGRAPH TYPE	PARAGRAPH TITLE
MODULE	&MODNAME Interpretation
PRIMITIVE ENTITY	GNAMEi /pe/ <Index Set Statement> Interpretation
COMPOUND ENTITY	GNAME <i> (Generic Calling Sequence) /ce/ <Index Set Statement> Interpretation
ATTRIBUTE	GNAME <i> (Generic Calling Sequence) /a or va/ <Index Set Statement> <Generic Range> Interpretation
FUNCTION OR TEST	GNAME <i> (Generic Calling Sequence) /f or t/ <Index Set Statement> Generic Rule; Intepretation

Figure 2.7 General Paragraph Notations

&SDATA There are some SOURCE DATA Concerning sources of supply.

PLANTi /pe/ There is a list of PLANTS.

SUP(PLANTi) /a/ {PLANT} :R+ Every PLANT has a SUPPLY CAPACITY Measured in units.

&CDATA There are some CUSTOMER DATA.

CUSTj /pe/ There is a list of CUSTOMERS.

DEM(CUSTj) /a/ {CUST} :R+ Every CUSTOMER has a non-negative DEMAND measured in units.

&TDATA There are some TRANSPORTATION DATA.

LINK(PLANTi, CUSTj) /ce/ Select {PLANT} x {CUST} covering {PLANT}, {CUST} There are some transportation LINKS from PLANTS to CUSTOMERS. There must be one LINK incident to each PLANT, and at least one LINK incident to each CUSTOMER.

FLOW(LINKij) /va/ {LINK} :R+ There can be a non-negative transportation FLOW (in units) over each LINK.

RATE(LINKij) /a/ {LINK} Every LINK has a TRANSPORTATION COST RATE for use in $/unit.

TOTAL COST(COST, FLOW) /f/ 1; SUMij (COSTij * FLOWij) There is a TOTAL COST associated with all flows.

T:SUP(FLOWi, SUPi) /t/ {PLANT}; SUMj (FLOWij) $\leq$ SUPi Is the total FLOW leaving a PLANY less than or equal to its SUPPLY CAPACITY? This is called the SUPPLY TEST.

T:DEM (FLOWj, DEMj) /T/ {CUST}; SUMi (FLOWij) = DEMj Is the total FLOW arriving at a CUSTOMER exactly equal to its DEMAND? This is called the DEMAND TEST.

Figure 2.8 Paragraph Schema: Transportation Model

- The generic calling sequence is the calling sequence of the genus. Notice that the genus symbolic indices are used here.

- The generic range provides the range of values associated with a genus element. This is similar to declaration statements in programming languages.

- The generic rule is the equation or rule used to assign a value to function and test genera.

- The interpretation is a narrative description of the genus or module. This is similar to a comment in programming code.

D. SUMMARY

Structured modeling provides the basis for a new generation, general purpose model management system. The model representations afforded by structured modeling provide views that are comprehensible to management, support the detail required by MS/OR personnel, and are computer executable. The graphical orientation of structured modeling allows for a computer-based implementation of a model management system with a state of the art user interface. The remainder of this thesis is concerned with the design of such an interface.

III. SYSTEM DESIGN

A. INTRODUCTION

Structured modeling establishes the foundation for a computer-based version of an operationally and functionally complete model management system (MMS). This thesis centers around the design and implementation of a graphical user interface to support this MMS. The prototyping development methodology was deemed the most practical for undertaking this project. The reasons behind this decision are presented in this chapter. System requirements specifications for the interface are also delineated and discussed. Finally, the overall system design is presented.

B. DEVELOPMENT METHODOLOGY

A prime consideration in the design and implementation of any computer-based system is the selection of an appropriate development methodology. Until recently, the traditional life-cycle (TLC) approach to system development was the only widely acceptable option. The deterrent to using the TLC approach is that adequate requirements specifications must be defined prior to proceeding with design and implementation. The difficulty here is that completely accurate requirements specifications are non-existent [Ref. 4: p. 57]. Therefore, determining when a specification adequately meets a user's

requirement is often a function of the deadline date for the TLC's analysis phase. This leads to the development of systems that are not what the user wanted nor what the analyst intended them to be.

> The demand for large, high quality software systems has increased to the point where a jump in software technology is needed. Rapid prototyping is one of the most promising methods proposed for solving this problem. [Ref. 5: p. 1]

Prototyping, in its purest form, is the development of a disposable version of the intended system used to supplement the analysis phase of the TLC. However, prototype systems that are well controlled and documented can shorten the TLC methodology by allowing a jump from the analysis phase directly into system implementation [Ref. 6: p. 252]. The issue, in this thesis, is not whether prototyping should or should not replace the TLC, but, that prototyping is a viable means of systems analysis that helps solidify the specification of a user's requirements.

The selection of an appropriate methodology is best made prior to the beginning of a project. An unspecified approach to system development can only lead to confusion and the project's eventual death. This decision is based on the characteristics of the proposed system. Systems with known costs and benefits and requiring little or no innovation are well-suited for the TLC methodology. The prototyping approach is most effective for systems containing new

technology, innovative software design, and uncertainty as to the operational benefits and costs [Ref. 6: p. 256].

The prototype methodology was adopted for use in this thesis for several reasons. The discipline of structured modeling is currently more theoretical than practical. As such, user requirements for the interface are very generalized in nature. A prototype system would allow the formulation of a more exact set of requirements specifications. Another factor in the selection of the prototype approach was to prove the feasibility of implementing the system in a micro-computer environment. The main concern here was that a micro-computer would not possess the speed or memory necessary to execute the system in real-time. The final factor was to show that a model management system could be effectively integrated with a database management system at the rudimentary level of model entry and review.

C. REQUIREMENTS SPECIFICATION

The initial phase in the prototyping methodology is to specify the system requirements. Unlike the TLC methodology, the prototyping methodology allows the definition of broad requirements specifications. The specifications become more refined as the prototype is constructed. This section presents the requirements considered essential for a graphical user interface to a model management system based on structured modeling.

The first requirement is that the system support real-time, graphical creation of structured models. The interface must provide a workspace for constructing both generic and modular structures. The workspace should allow assignment of meaningful names or labels to the various model elements.[1] Relationships between the model elements will be captured with directed line segments (or arcs) as stipulated in the concepts of structured modeling.

The system should verify that the structure is allowable under the constraints imposed by structured modeling. If a generic structure is being created, the system ought to ensure that the model is acyclic and that there are no improper calling sequences (i.e., a primitive entity calling an attribute or a variable attribute calling a function). In a modular structure, the system needs to detect non-terminal model elements that are not modules and terminal model elements that are not genera. The intent is to verify adherence to the rules of structured modeling, not the accuracy or completeness of a model.

The interface needs to allow the entrance of generic and module paragraph data. This portion of the interface must only request information from the user that cannot be extrapolated from the graphical representation. Paragraph

---

[1]For the remainder of this thesis, the phrase "model elements" will refer to genera and modules. The phrase "elements of a model" will connote the element definition given in the structured modeling chapter.

entries are then checked for syntactical errors and missing information. Upon completion of model construction, all data is assimilated, placed in the proper database format, and written to external memory.

Another major specification for the system is the capability to generate genus graphs and modular structures from a database representation. The database representation is to have no references to graphical information. Therefore, the system must determine where to place model elements and the arcs representing the calling sequence segments. The goal is to re-create a functionally correct representation of the model that is aesthetically acceptable. The format for the graphical representations should be the same as that described for the creation of genus and module structures.

In conjunction with the re-creation of graphical representations, is the requirement to view model information not captured graphically. This feature should allow the user to select a model element for which the corresponding paragraph data is then presented in a clear and concise format.

The third major system specification for the interface is the capability to edit a model. Model editing is to be allowed during the creation of a model and after its recall from a database. There should be a means to delete and add model elements. If a model element is deleted, the calling sequence must also be deleted and the user prompted as to the disposal of the subordinate nodes. If an element is added,

the user needs to be prompted for the entry of the element's paragraph data. Editing of text within a paragraph description should be allowed without affecting the graphical representation.

A complete interface will also support miscellaneous structured modeling presentations. The user should be allowed to create specialized views of the genus graphs. Adjacency and reachability matrices are also a necessary part of the interface.

The system requirements specifications just discussed are the starting point for the design and implementation of the prototype system. Under the TLC methodology, these requirements would not be adequate for proceeding into the design phase. A large amount of additional time and effort is still required to define more exact specifications in either methodology. However, by prototyping the system, progress can be made on the implementation as specifications are better defined.

D. PROTOTYPE DESIGN

The design and implementation of the entire system described in the requirements specifications was not undertaken as part of this thesis. O'Dell [Ref. 7] conducted concurrent research concerning the on-line creation and editing of structured models and their subsequent entry into a database. The work presented in this thesis addresses the re-creation of model structures from the database

representation. The capability of viewing module and genus graphs is also discussed.

1. Database Design

A relational database representation of a structured model was adopted for this system. The database design was developed by Dolk [Ref. 3]. The basic premise of this design is that a structured model can be fully represented with two relations: the **ENTITY** relation and the **RELSHIP** relation. The **ENTITY** relation provides the description of a single model element. The **RELSHIP** relation allows representation of cross-referencing between model elements. The specific structures of these relations are presented in Figures 3.1 and 3.2.

The **ENTITY** relation uses ENAME and ETYPE as the key for referencing a particular record. ENAME, ETYPE, IDX, IDX_STMT, GRANGE, GRULE, and COMMENTS are the **ENTITY** relation fields necessary to store model element paragraph information. The DNAME, DATE_ADDED, LAST_MOD, and NMODS fields are included for the administration of a multi-user model management system. Model elements not requiring all the information in the **ENTITY** relation leave the appropriate fields null (i.e., GRANGE and GRULE are null for a primitive entity representation). A SQL SELECT command is used to extract the correct tuple for a selected model element.

The **RELSHIP** relation uses the RTYPE, E1NAME, E1TYPE, E2NAME, E2TYPE, and REL_POS fields for representing the

FIELD NAME	FIELD TYPE	FIELD SIZE	DESCRIPTION
ENAME	CHAR	12	Abbreviated model element name
ETYPE	CHAR	8	Model element type
DNAME	CHAR	30	Descriptive model element name
DATE_ADDED	DATE		Date when model element was added to the database
LAST_MOD	DATE		Date when model element information was last updated or modified
NMODS	NUMBER	5	Number of modifications
IDX	CHAR	4	Symbolic genus index
IDX_STMT	CHAR	100	Index set statement
GRANGE	CHAR	20	Generic range
GRULE	CHAR	100	Generic rule
COMMENTS	CHAR	100	Model element interpretation

Figure 3.1 **ENTITY** Database Relation

FIELD NAME	FIELD TYPE	FIELD SIZE	DESCRIPTION
RTYPE	CHAR	8	Type of relationship (Calls or Contains)
E1NAME	CHAR	12	Calling element name
E1TYPE	CHAR	12	Calling element type
E2NAME	CHAR	12	Called element name
E2TYPE	CHAR	12	Called element type
ACC_METHOD	CHAR	12	Access method
FREQ	NUMBER	6	Access frequency
REL_POS	NUMBER	6	Position of heirarchy for monotone ordering

Figure 3.2 **RELSHIP** Database Relation

relationships between model elements in generic structures and modular structures. The model element described by E2NAME and E2TYPE is subordinate to the model element referred to by E1NAME and E1TYPE. The ACC_METHOD and FREQ fields are used for multi-user MMS administration. RTYPE, E1NAME, E1TYPE, E2NAME, and E2TYPE form the composite key for the **RELSHIP** relation. The RTYPE field holds either the value "contains" or "calls." An RTYPE value of "contains" refers to a modular structure relationship and a value of "calls" refers to a generic structure relationship.

Views of the **ENTITY** relation are defined for the various model element types. The view of each model element type contains the paragraph entries for that type as delineated in Figure 2.7. The SQL commands establishing these views are as follows:

- For the primitive entity view: CREATE VIEW PE as SELECT DNAME, ENAME, DATE_ADDED, LAST_MOD, NMODS, IDX, COMMENTS from ENTITY where ETYPE = 'PE';

- For the compound entity view: CREATE VIEW CE as SELECT DNAME, ENAME, DATE_ADDED, LAST_MOD, NMODS, IDX, IDX_STMT, COMMENTS from ENTITY where ETYPE = 'CE';

- For the attribute view: CREATE VIEW CE as SELECT DNAME, ENAME, DATE_ADDED, LAST_MOD, NMODS, IDX_STMT, GRANGE, COMMENTS from ENTITY where ETYPE = 'ATT';

- For the variable attribute view: CREATE VIEW VA as SELECT DNAME, ENAME, DATE_ADDED, LAST_MOD, NMODS, IDX_STMT, GRANGE, COMMENTS from ENTITY where ETYPE = 'VA';

- For the test view: CREATE VIEW TEST as SELECT DNAME, ENAME, DATE_ADDED, LAST_MOD, NMODS, IDX_STMT, GRULE, COMMENTS from ENTITY where ETYPE = 'TEST';

- For the function view: CREATE VIEW FCN as SELECT DNAME, ENAME, DATE_ADDED, LAST_MOD, NMODS, IDX_STMT, GRULE, COMMENTS from ENTITY where ETYPE = 'FCN';.

Views are also defined to capture the genus and module relationships represented in the **RELSHIP** relation. The SQL commands for the two views are:

- For the 'calls' relationship: CREATE VIEW CALLS as SELECT E1NAME, E1TYPE, E2NAME, E2TYPE from RELSHIP where RTYPE = 'calls';

- For the 'contains' relationship: CREATE VIEW CONTAINZ as SELECT E1NAME, E1TYPE, E2NAME, E2TYPE from RELSHIP where RTYPE = 'contains';.

2. Control Structure

After establishment of the database representation, the design of the interface was undertaken. The first consideration in this process was the construction of a suitable control structure. This is required for proper operation of the prototype.

A control structure is necessary that allows the user to access the various features of the interface. A hierarchical control structure provides this ability. At the first level, the user is given the option of selecting from model creation, recall, and editing as well as miscellaneous modeling and system[2] functions. The second level commands are those required in execution of the upper level task. This control structure allows the incorporation of new features in as many levels as dictated functionally.

---

[2]System functions include directory changes, file deletion, renaming of files, etc.

3. Screen Design

The screen format chosen reflects the design of the control structure. It is an adaptation from current commercial software products such as Lotus 1-2-3. There is a row of text across the top of the screen displaying information concerning the model being displayed in the workspace. A row of text across the bottom of the screen provides a quick reference to various program commands. The model workspace is the area between the two rows of text.

To accomplish the required screen display, a world coordinate system is first instituted. There are two advantages to using a world coordinate system [Ref. 8: p. 38]. First, the system refers to its own graphical coordinates rather than screen coordinates. This means that the system will be transferrable between display devices regardless of their resolution. The second advantage is that all graphics is scaled to the size of the viewport. The biggest disadvantage of this approach is that the Halo graphics package does not allow text to be scaled in relation to the viewport.

The screen construction begins by initially displaying the required rows of text. A viewport is then established which is the same size as the workspace. This allows refreshing, changing, or clearing the workspace without rewriting the rows of text.

4. Genus Icons

The next design concern was the establishment of appropriate graphical depictions of the various model elements in a genus graph. Geometrically shaped and uniquely colored icons are logical choices for this purpose. When combined with a textual label, there exist three visual means of identifying a particular model element. These icons are identically sized for easier positioning within the work-space.

An additional concern in the design of the genera icons is support for the graphical representation of generic calling sequence segments. In structured modeling, all genus types (except primitive entities) have a calling sequence. Therefore, the icons include an arrow that points to the geometric shape (the primitive entity icons omit the arrow). The calling sequence arcs are drawn from the top of the geometric shape of the called element to the tail of the arrow of the calling element. The icons for the various genus types are shown in Figure 3.3.

5. Drawing Generic Graphs

With the workspace and graphical representation of model elements defined, the next step is to define an algorithm for the positioning of model elements in a genus graph. The basis for this algorithm is that a genus graph can be divided into layers or levels. On the base level are all the genera that have no calling sequences or the

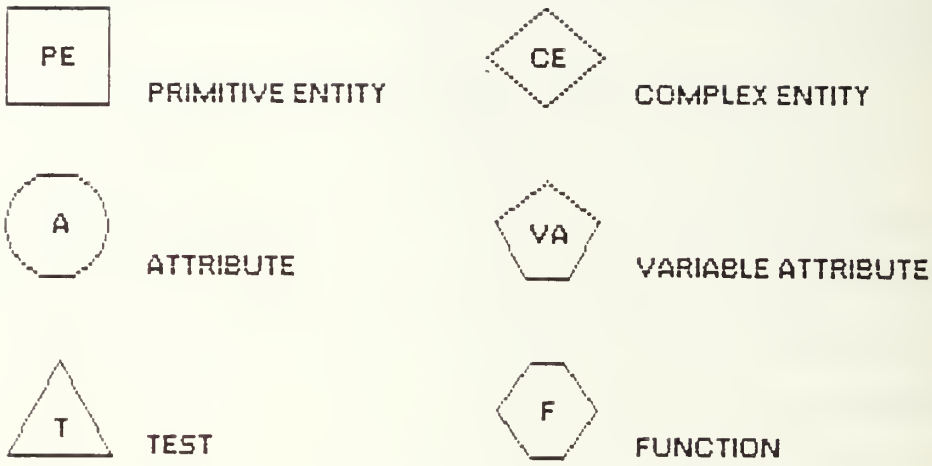


Figure 3.3 Graphical User Interface Genus Icons

primitive entities. The second level consists of the genera that call the primitive entities. The third level genera call second level elements and so on. The y-coordinate for an icon position is based on the level of the corresponding genus.

It is possible for a genus to call subordinate genera from two different levels.[3] If this occurs, a placeholder is included on a level. This is to allow a calling sequence arc to be drawn across a level without interfering with the genus icons. The placeholder is the same size as the genera icons. Graphically it is a single vertical line that allows arcs to span levels.

The localization of related model elements is necessary to enhance the aesthetic quality of the graphical representation. Localization of model elements is the process of grouping directly related genera in the same area of the workspace. This is desirable so that the arc length and arc intersection are kept at a minimum. Arcs that are drawn the length of the screen are functionally correct, but greatly degrade user comprehension of the model.

The positioning algorithm requires that the model elements be manipulated based on name, generic type, and the level that they occupy. Therefore, institution of an internal data structure was necessary to support management

---

[3]This situation can occur in several instances. For example, a test element that calls an attribute and a primitive entity like T:DEMAND in the transportation model.

of elements for an entire model. The positioning information for an entire model is represented through an array of records. The position of individual model elements is depicted in a record. Each record contains a field for the following entities: the element name, the element type, the level, the icon's x-coordinate, and the icon's y-coordinate. Henceforth, this array will be referred to as the position array. The model element names and types are extracted from the database using the following SQL command:

- SELECT ENAME, ETYPE from ENTITY;.

Additionally, the positioning algorithm needs information concerning the cross-referencing between the genera. Again, an array of records was selected as the appropriate internal data structure. Each record in this array represents a separate generic relationship. There exists a record for each arc in the generic graph. The records contain fields for the calling element name, calling element type, called element name, and called element type. This array is called the relation array. The SQL command to recall this information is:

- SELECT E1NAME, E1TYPE, E2NAME, E2TYPE from CALLS;.

Once the relative position of the genera are established, the x-coordinates are assigned so that the model is centered in the workspace. Appendix A contains the mini-specification for the complete genus graph positioning algorithm.

The next step in drawing the genus graph is drawing the arcs between the genera. The algorithm bases the drawing of lines on the relationships between the genera. There are no references to the length or endpoints of a line. The algorithm references the position of an icon to determine the endpoints of a line. As both the positioning of elements and line drawing are based on algorithms, the genus graph representation is totally independent of its mode of creation. This allows a user to create a genus graph by direct entry of model information into the database. However, if the model was created graphically, the algorithm should provide a representation that closely resembles the original genus graph.

The drawing of lines is accomplished by stepping through the relation array one record at a time. The calling element and called element are determined. A search is then made of the position array to determine the x- and y-coordinates of the respective icons. A line is drawn between the two icons with the line's endpoints being a function of the icon's x- and y-coordinates. A line that passes through a level of model elements must avoid the icons on that level. This is accomplished by drawing the line in two segments. The first line segment is drawn from the called element to a placeholder. The second segment is drawn from the placeholder to the calling element.

6. Drawing Modular Structures

The purpose of this routine is to represent the modular structure exactly as shown in Figure 2.4. The modular structure is in actuality a simple tree structure. As such, the number of terminal nodes (or genera) represents the structure's width. Therefore, the width of a modular structure is determined by counting the number of genera. This information is used to center the modular structure vertically within the workspace.

Physical reconstruction of the modular structure begins by placing the root module at the vertical center of the workspace. The model elements contained directly within this root module are positioned in a second column. Each column represents a level of hierarchy in the module structure. The positioning of these second column model elements is based on the eventual vertical position of the terminal genera. The entire structure is built in this manner, level by level.

The requirement for monotone ordering of a modular structure mandates that a genus within a module make no forward references to other genera within the module. This means that there is a definite order to the vertical position of the genera. The relative position of the genera is included in the database representation of the **RELSHIP** relation. This information is referenced prior to physically positioning the genera of a module with the SQL command:

- SELECT E1NAME, E1TYPE, E2NAME, E2TYPE, REL_POS from CONTAINZ;.

7. Displaying Model Element Paragraphs

The final system capability implemented as part of this thesis is module and genus paragraph viewing. When this function is selected by the user, a graphics cursor is enabled. The user moves the cursor about the genus graph or modular structure using the cursor control keys. The cursor is positioned over the desired model element and the RETURN key depressed to display the corresponding paragraph.

A database query is made to access the paragraph information as it is not stored internal to the program. The SQL command for acquiring the paragraph information is:

- SELECT IDX,IDX_STMT,GRANGE,GRULE,COMMENTS from ENTITY where ENAME = '<selected model element name>' and ETYPE = '<selected model element type>';.

Some of the retrieved fields will be NULL depending on the model element type. The paragraph data is then formatted and displayed within a window placed over the workspace. When the user completes viewing the paragraph, the ESCAPE key is depressed. This restores the image beneath the window and returns cursor control to the user.

E. CONCLUSION

Generalized requirements of a graphical user interface for a model management system were presented in this chapter. Also discussed were the design considerations for the system. It should be noted that much of the design work was completed

as the interface was being implemented. The prototyping development strategy provides the flexibility for easy modification of requirements during design and implementation.

IV. PROTOTYPE EVALUATION

A. INTRODUCTION

The design and implementation of the prototype graphical user interface for a model management system was discussed in Chapter III. In this chapter, the evaluation of that prototype is discussed. The areas of evaluation are suitability of the programming environment, adequacy of the initial requirements specifications and compatibility with other research efforts. The evaluation results include refined system requirements and a recommended disposition of the prototype.

B. SUITABILITY OF PROGRAMMING ENVIRONMENT

Originally there was some hesitancy in the selection of the programming environment described in Chapter I. First, the graphics facilities of the IBM PC were thought to be inadequate for an effective implementation of the graphical user interface. Attainment of acceptable response times was also a major concern in using the IBM PC. Secondly, the author had no experience using either the C programming language or the ORACLE database management system. Finally, the Halo graphics package, while highly touted commercially, was an unknown commodity. Despite these initial concerns, the prototype implementation uncovered no programming environment deficiencies.

The stipulated hardware configuration and the Halo graphics package far exceeds original expectations. The requirement for the enhanced graphics adapter (EGA) proved a wise decision. It allows presentation of an aesthetically pleasing display with a complete IBM PC color palette and normally sized text characters. The hardware response time is also more than adequate. A genus graph filling the entire on-screen workspace, is cleared and redrawn in less than a second.

The Halo graphics package provides two-dimensional graphics functions similar in capability to more specialized graphics systems such as the Silicon Graphics' IRIS-2400 work station. The more advanced Halo functions used in the interface program include world coordinate system definition, viewport definition, polygon filling, and rubberband boxes[4]. The Halo version used was fully compatible with the Lattice-C compiler and supported numerous graphics drivers.

The Lattice-C compiler provides the programming flexibility that the system requires. At one end of the spectrum, it permits integration with the ORACLE DBMS and the Halo graphics package. At the other end, direct calls to the MS-DOS BIOS interrupts are allowed. The only criticism of the

---

[4]The rubberband box function, when called initially, draws a box at a specified location. Subsequent calls will cause the first box to be erased and a new box to be drawn at whatever position is specified. A rubberband box was used as the graphics cursor in selecting model elements for displaying paragraphs.

Lattice-C compiler is that a source level debugger is not included as part of the basic software package.

Part of the complete PC ORACLE DBMS software package are library modules which are linkable with Lattice-C produced modules. These modules allow use of ORACLE and SQL (Standard Query Language) commands from within the C source code. Once the system is compiled and linked, execution relies on the ORACLE system being embedded within the IBM PC's extended memory.

The overall performance of the graphical user interface in this programming environment is deemed sufficient. However, consideration should be given to implementations utilizing other hardware and software configurations.

The ultimate goal for the complete MMS is integration within a corporate information system. With this in mind, a mainframe version of the interface is highly desirable. This provides two options for hardware implementation. The first is using the mainframe's remote terminals for displaying and manipulating the interface. The second is using micro-computers as front-end processors for the mainframe MMS.

The development of the interface could be greatly accelerated if an existing software product could be adapted to meet the system requirements. Apple's HyperCard provides a graphical database environment that might be well suited for this application. A factor in the decision to build

either a new or a "turn-key" system is whether it can be integrated with other parts of the MMS.

C. REQUIREMENTS ENHANCEMENTS

The final step in the prototyping design strategy is evaluating the system for both satisfaction of user requirements and recommendations for further implementation. This section is an analysis of the prototype's weaknesses and deficiencies in the area of user requirements. Refinements to the system design are suggested to rectify the prototype imperfections. The main purpose of the prototype was to prove the feasibility of re-creating adequate structured modeling representations from a database representation. As such, little attention was given to how the user communicates with the system. Several of the interface functions can be made more "user friendly" by improving the human-computer interaction.

Currently, selection of system features use the function keys (F1, F2, F3, etc.). However, this method of selection is not in keeping with the spirit of a completely modern and graphical interface. Research shows that the preferred picking mechanism is a mouse [Ref. 9: p. 108]. The user would use the mouse to select from various pull-down menu options. The desired function is then selected from the pull-down menu. The incorporation of the mouse requires no major control structure modifications. New software is

unnecessary as mouse functions are fully supported by the Halo graphics package.

A mouse could also enhance the selection of model elements when displaying genus and modular paragraph information. The mouse cursor is placed on a model element's icon and selection made by depressing the mouse button. The system then reads the mouse coordinates and searches the array containing icon positions to determine the model element selected. This information is then used to make the appropriate database query.

Another weakness of the system concerns the drawing of arcs representing relations between model elements. The current procedure allows arcs to span levels[5] of the generic structure, providing a functionally correct representation. However, the aesthetic quality of the display is greatly diminished using this procedure. A better procedure for drawing arcs would certainly enhance the quality and understandability of the generic representation. This procedure needs to eliminate the line disjointedness created by the placeholder icon used in the present procedure.

The single screen workspace provided in the prototype was found to be too small for the representation of many models. One option is to use smaller icons for the model elements. The difficulty of this solution is that the text cannot be

---

[5]A complete discussion on the level structure of genus graphs is given in Chapter three.

scaled in proportion to the icons and in effect places a lower limit on the size of the icons. The second, and more feasible, option is a virtual workspace. The user can use the arrow keys to scroll left, right, up, or down throughout the model representation. In addition, the system should provide the capability of proceeding directly to a model element specified by name.

The improvements discussed will greatly increase the user friendliness and overall quality of the prototype. At this point, however, a decision must be made to continue improving the prototype or use the prototype as the input to the design phase of the traditional life-cycle (TLC). Recommendations concerning this decision will now be presented.

D. DISPOSITION OF THE PROTOTYPE

Transformation of a prototype system into a production system is currently a highly debated topic. Advocates of the prototyping methodology contend that the TLC process is unnecessarily long and that prototyping is the solution to timely software production [Ref. 6: p. 252]. Defenders of the TLC approach view prototypes as 'toy' systems that aid in fine tuning the overall system design [Ref. 4: p. 226], [Ref. 10: p. 3]. The characteristics of the prototype graphical user interface support the latter opinion.

The impetus behind the development of the graphical user interface prototype was to produce a working portion of the complete system. Therefore, several features considered

essential in a production system were omitted. Among these features are extensive error-checking mechanisms, source code optimization, and well documented trails of design and implementation decisions [Ref. 4: p. 226].

Testing the prototype focused on the correctness of the various interface displays rather than the identification of program defects. Therefore, numerous minor bugs remain uncorrected. Optimization of source code was limited to generation of commonly used modules. The only available documentation for the system is comments within the source code and this thesis. There is no method to trace the development strategy in converting the design into source code. The institution of programming standards is not easily accomplished during the maintenance phase. Thus, using the prototype as the basis for proceeding on to the TLC design phase will provide the potential to make improvements in these essential areas.

A second factor prompting the decision to forego the prototype is the need to integrate this portion of the graphical user interface with other research efforts. As stated in Chapter III, the model creation and editing functions were designed and implemented concurrent with this thesis. Integration of the two prototypes will require the resolution of the following design issues:

- Determination of data structures satisfying the needs of model building and re-creation functions.

- Development of a screen design supporting all the various functions of the complete system.
- Design of a control structure to support the entire graphical user interface.
- Model element icons standardized on the basis of size, shape, color, and labeling.
- Standardization of the model element paragraph displays to facilitate entry, modification, and viewing.
- A method of re-creating a model representation that exactly resembles the representation as it was originally created.

Resolving the differences in representation of the model as created and re-created is the most difficult design issue. The current model creation program allows the user to place icons and draw relational arcs arbitrarily while the re-creation program draws the icons and arcs algorithmically. This results in considerable disparities between the created (Figure 4.1) and re-created (Figure 4.2) representations. There are two possible solutions to this problem. The first is to expand the database to include a relation for storing graphical coordinates for the icons and arcs. The second solution is to reformat the user's representation into the algorithmic representation each time the user represents a relation between the icons. The advantage of the first solution is that the user obtains a representation exactly as drawn. The disadvantage is the need for additional database storage space. The second is a trade-off in that the user's exact model representation is forfeited in favor of saving storage space. The decision as to which method is better,

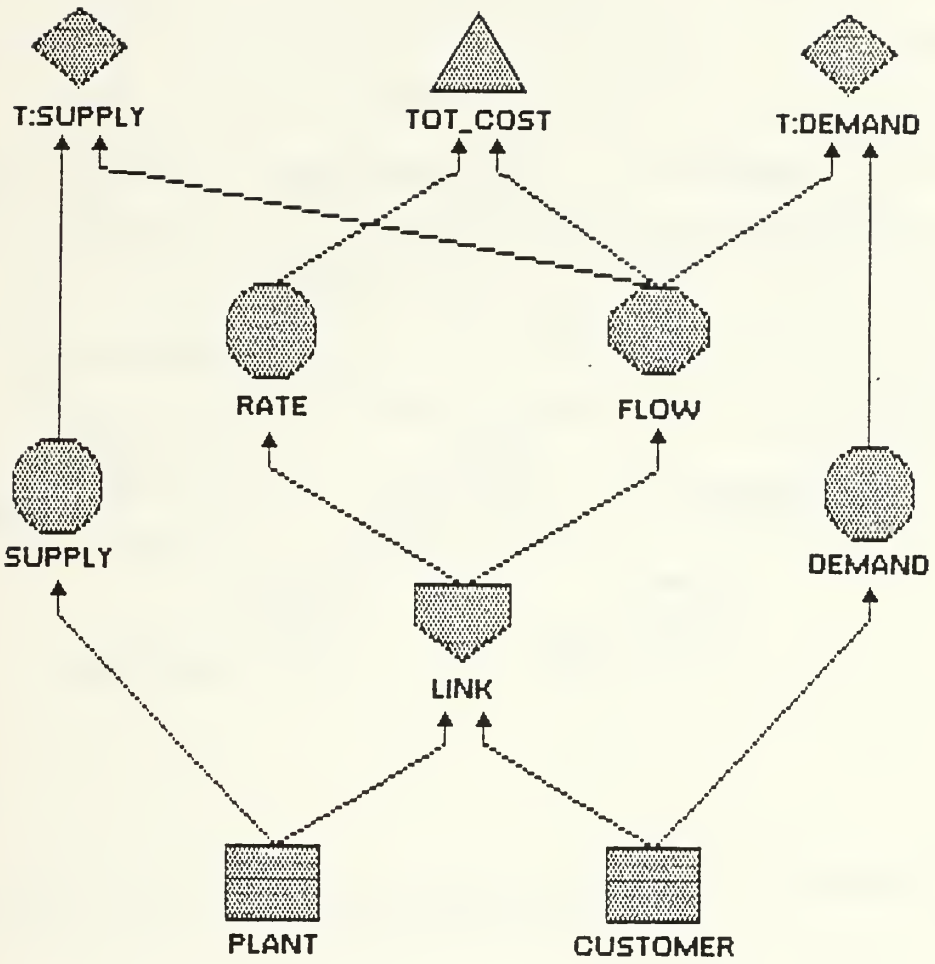


Figure 4.1 Graphical User Interface Genus Graph: Created Version

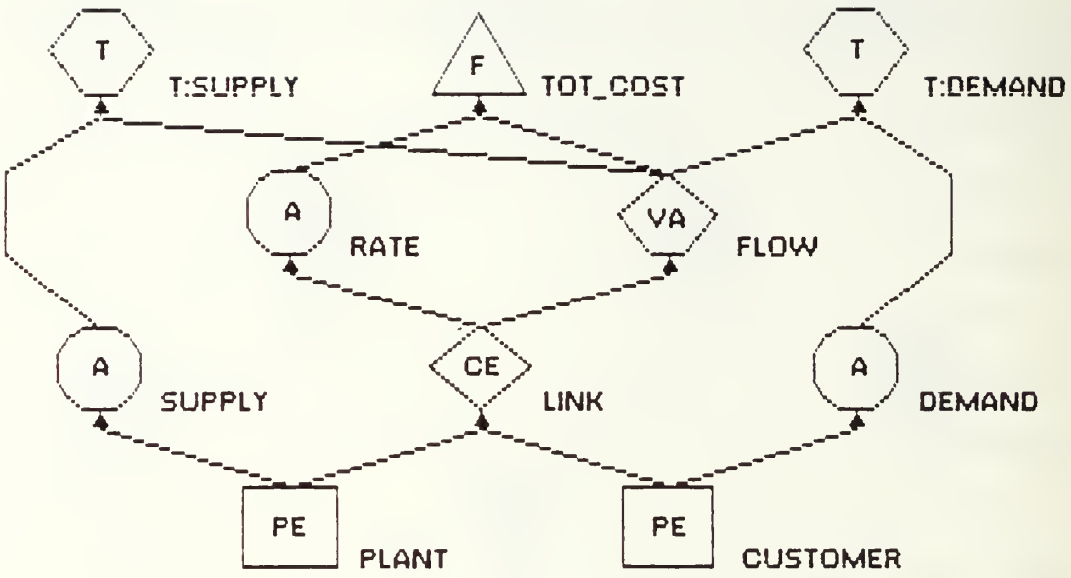


Figure 4.2 Graphical User Interface Genus Graph: Re-created Version

can best be made through a complete life-cycle analysis of the problem.

To reiterate, design issues are extremely difficult to correct in a maintenance environment. Integration of the two systems is mainly a maintenance effort. In addition, the final system must include several functions not prototyped. These factors strongly support the decision of beginning the design of the system anew using the traditional life cycle.

In conclusion, the prototype, while not acceptable as a production system, allowed highly productive system analysis. User requirements were solidified and a basis for the design phase of the production system was provided. Traditional life-cycle development is, therefore, greatly enhanced through the use of a prototype.

V. CONCLUSION

A. DISCUSSION

This thesis was conducted to prove the feasibility of implementing a graphical user interface for a model management system (MMS). Research in this area is necessary in overcoming the major obstacles confronting the management science and operations research (MS/OR) disciplines. The low productivity of MS/OR personnel and the low level of model acceptance by management are the two largest factors hindering the growth of modeling technologies. An MMS that is easy to use and understand will greatly aid in overcoming these difficulties.

A new type of MMS must possess several features to gain acceptance by both management and MS/OR practitioners. It should use a single model representation for all levels of analysis. The representation must be general enough to support numerous types of models. The system must also support the entire modeling life-cycle, allow implementation on a micro-computer, support integration with a database management system (DBMS), and provide fast, accurate results. Structured modeling is a framework for construction of such a system.

A significant part of this MMS will be the user interface. Structured modeling relies on pictorial

representations of models suggesting the use of a graphical interface. This thesis delineates an initial set of requirements specifications for the graphical user interface. The completeness of the specifications could not be verified thus warranting the use of a prototype development strategy. The portion of the interface undertaken in this thesis concerns the re-creation of graphical model representations from a database representation of the model.

Prototyping allowed rapid design and implementation of this part of the graphical user interface. The system used an IBM PC with an embedded ORACLE DBMS. It proved the feasibility of implementing a modern user interface for a structured modeling based MMS. The prototype was also evaluated to determine the completeness of the user specifications and the disposition of the prototype. The evaluation of the prototype revealed several useful functions overlooked in the initial specification of requirements. However, it was recommended that the prototype not be enlarged to incorporate these improvements. Instead, the prototype and recommended enhancements should be the basis of design for conducting a complete life-cycle implementation of the complete user interface.

B. RECOMMENDATIONS FOR FUTURE STUDY

Two research opportunities in the area of computer-based model management systems surfaced as a result of this thesis. The first concerns consolidation with concurrent research

efforts. The need exists to integrate the refined requirements specifications from the various efforts into a single and complete interface.

A second area of interest is the commercial viability of the entire MMS. Since users interact with a system through its interface, the graphical user interface is an excellent tool for evaluation in this area. Model practitioners and management can use the interface to provide inputs concerning the demand for such a product.

Research in the area of model management should focus on increasing the organizational acceptance of MS/OR models. The complexity of many decision making processes is greatly reduced through the use of models. Therefore, a model management system should be an integral part of an organization's information system. Neither the modeling ideology used, the user interface employed, nor the system implementation environment should inhibit the attainment of the actual goal of increased model usage.

APPENDIX A

SELECTED MINI-SPECIFICATIONS

1 GENUS GRAPH POSITIONING ALGORITHM

1.1 Initialize the level counter to 1.

1.2 For each record in the relation array:

1.2.1 Determine if the called element is a primitive entity.

1.2.1.1 If so, enter the called element name, called element type, and value of the level counter into the next available record in the position array and then proceed to step 1.2.2.

1.2.1.2 If not, proceed to step 1.2.2.

1.2.2 Loop back to step 1.2.1 until all relation array records have been checked.

1.3 Increment the level counter by 1.

1.4 Eliminate all duplicate records in the position array.

1.5 For each record in the relation array:

1.5.1 Determine if the calling element calls an element on the level below the current level counter value.

1.5.1.1 If so, enter the calling element name, calling element type, and value of the level counter into the next available record in the position array and then proceed to step 1.5.2.

1.5.1.2 If not, proceed to step 1.5.2.

1.5.2 Loop back to step 1.5.1 until all relation array records have been checked.

1.5.3 For each record in the position array with a a level value equal to the current level counter value:

1.5.3.1 Determine if the calling element calls another element on the same level.

1.5.3.1.1 If so, replace all occurrences of the element with placeholders and proceed to step 1.5.3.2.

1.5.3.1.2 If not, proceed to step 1.5.3.2.

1.5.3.2 Loop back to step 1.5.3.1 until all position array records have been checked.

1.5.4 Eliminate duplicate records in the current level.

1.5.5 Increment the level counter by 1.

1.5.6 Loop back to step 1.5.1 until all calling elements in the relation array have been transferred to the position array.

1.6 Determine the level with the most elements.

1.6.1 Use the value from step 1.6 to center the graph horizontally and calculate the x-coordinates.

1.7 Determine how many levels in the graph.

1.7.1 Use the value from step 1.7 to center the graph vertically and calculate the y-coordinates.

2 GENUS GRAPH LINE DRAWING ALGORITHM

2.1 For each record in the relation array:

2.1.1 Determine the calling element name, calling element type, called element name, called element type and the level value for each element.

2.1.2 Get x- and y-coordinates for the elements from the position array.

2.1.3 If the difference between the level values equals 1:

2.1.3.1 Calculate the starting and ending points for the line.

2.1.3.2 Draw the line.

2.1.4 If the difference between the level values is greater than 1:

2.1.4.1 For each level crossed:

2.1.4.1.1 Search the position array for the placeholder icon

that represents the level crossing for that relation and calculate line starting and ending points.

2.1.4.1.2 Draw the line.

2.1.4.2 Calculate the starting and ending points for the line from the last placeholder icon to the calling

element icon.

2.1.4.3 Draw the line.

2.1.5 Loop back to step 2.1.1 until all records have been checked.

APPENDIX B

GRAPHICAL USER INTERFACE SOURCE CODE

```
/*************************************************************
 *  HEADER FILE: GENUS.H                                     *
 *  WRITTEN BY: MARVIN A. WYANT, JR.                         *
 *  DATE OF LAST MODIFICATION: 11 MARCH 1988                 *
 *  PURPOSE: THE PURPOSE OF GENUS.H IS TO ASSIGN VALUES      *
 *           TO GLOBAL CONSTANTS AND GLOBAL DATA             *
 *           STRUCTURES.                                     *
 *************************************************************/

/*  DEFINE GLOBAL CONSTANTS FOR THE COLORS  */

#define BLACK 0
#define BLUE 1
#define GREEN 2
#define CYAN 3
#define RED 4
#define MAGENTA 5
#define BROWN 6
#define LT_GRAY 7
#define DARK_GRAY 8
#define LT_BLUE 9
#define LT_GREEN 10
#define LT_CYAN 11
#define LT_RED 12
#define LT_MAGENTA 13
#define YELLOW 14
#define WHITE 15
#define NOTHING -1

/*  DEFINE GLOBAL CONSTANTS FOR THE FUNCTION KEYS  */

#define F1 59
#define F2 60
#define F3 61
#define F4 62
#define F5 63
#define F6 64
#define F7 65
#define F8 66
#define F9 67
#define F10 68
```

```
#define SHIFT_F1 84
#define SHIFT_F2 85
#define SHIFT_F3 86
#define SHIFT_F4 87
#define SHIFT_F5 88
#define SHIFT_F6 89
#define SHIFT_F7 90
#define SHIFT_F8 91
#define SHIFT_F9 92
#define SHIFTF10 93

#define CTRL_F1 94
#define CTRL_F2 95
#define CTRL_F3 96
#define CTRL_F4 97
#define CTRL_F5 98
#define CTRL_F6 99
#define CTRL_F7 100
#define CTRL_F8 101
#define CTRL_F9 102
#define CTRL_F10 103

#define ALT_F1 104
#define ALT_F2 105
#define ALT_F3 106
#define ALT_F4 107
#define ALT_F5 108
#define ALT_F6 109
#define ALT_F7 110
#define ALT_F8 111
#define ALT_F9 112
#define ALT_F10 113

/*  DEFINE GLOBAL CONSTANTS FOR THE CURSOR KEYS  */

#define HOME 71
#define UP_ARROW 72
#define PG_UP 73
#define LEFT_ARROW 75
#define RIGHT_ARROW 77
#define END 79
#define DOWN_ARROW 80
#define PG_DN 81

#define CTRL_HOME 119
#define CTRL_PG_UP 132
#define CTRL_LEFT_ARROW 115
#define CTRL_RIGHT_ARROW 116
#define CTRL_END 117
#define CTRL_PGDN 118
```

```
#define INS 82
#define DEL 83
#define ESC 27

/*  A GLOBAL ARRAY FOR STORING THE SCREEN WHEN USING A
    WINDOW */

int window_array[30000];

/*  TYPE DEFINITION FOR THE DATA STRUCTURE TO REPRESENT
    RELATIONS BETWEEN MODEL ELEMENTS */

typedef struct { char e1name[9];
                 char e1type[3];
                 char e2name[9];
                 char e2type[3];
                 int rel_pos;
               } RELSHP;

/*  TYPE DEFINITION FOR THE DATA STRUCTURE TO REPRESENT THE
    POSITION OF A MODEL ELEMENT */

typedef struct { char name[9];
                 char type[3];
                 float xpos;
                 float ypos;
                 int level;
               } POSIT;

/*************************************************************
                        END GENUS.H
 *************************************************************/
```

```
/*************************************************************
 *  PROGRAM FILE: MAIN.C                                     *
 *  WRITTEN BY: MARVIN A. WYANT, JR.                         *
 *  DATE OF LAST MODIFICATION: 11 MARCH 1988                 *
 *  PURPOSE: CONTAINS THE MAIN CONTROL STRUCTURE FOR THE     *
 *           MODEL MANAGEMENT SYSTEM GRAPHICAL USER          *
 *           INTERFACE.  THIS MODULE ALSO ALLOTS SPACE       *
 *           FOR THE ARRAY OF DATA STRUCTURES.               *
 *************************************************************/


#include "stdio.h"     /* A STANDARD LATTICE-C HEADER FILE */
#include "string.h"    /* A STANDARD LATTICE-C HEADER FILE */
#include "stdlib.h"    /* A STANDARD LATTICE-C HEADER FILE */
#include "genus.h"     /* DEFINES PREPROCESSOR VARIABLES,  */
                       /* TYPES, AND DATA STRUCTURES       */
#include "utils.h"     /* ADDITIONAL C UTILITIES NOT       */
                       /* INCLUDED IN LATTICE-C            */
#include "icons.h"     /* FUNCTIONS THAT DRAW THE VARIOUS  */
                       /* ELEMENT ICONS                    */
#include "initdisp.h" /* INITIALIZES THE GRAPHICS SYSTEM  */
                       /* AND DRAWS THE SYSTEM SCREE       */
#include "help.h"      /* PRESENTS THE HELP SCREENS        */
#include "drawgen.h"   /* CREATES A GENUS STRUCTURE FROM   */
                       /* THE ELATIONS DESCRIBED BETWEEN   */
                       /* THE MODEL ELEMENTS               */

/* THE MAIN CONTROL STRUCTURE */

main()
   {
    /* HOLDS THE VALUE OF THE KEY INPUT BY THE USER */
    int key;

    /* ALLOTS SPACE FOR THE DATA STRUCTURES */
    RELSHP *relptr;
    POSIT *posptr;
    POSIT *temptr;

    relptr = (RELSHP *) calloc(100,sizeof(RELSHP));
    posptr = (POSIT *) calloc(500,sizeof(POSIT));
    temptr = (POSIT *) calloc(500,sizeof(POSIT));

    /* INITIALIZE THE GRAPHICS SYSTEM AND THE SYSTEM
       SCREEN */
    initdisplay();
```

```
while (1)
   {
    /* GET THE USER'S INPUT */
    key = getkey();

    switch ( key )
       {
        /* DISPLAY HELP SCREENS */
        case F1: help();
                 break;

        /* RE-CREATE A GENUS STRUCTURE */
        case F2: draw_genus(relptr,posptr,temptr);
                 break;

        /* SPARE INPUT KEYS */

        case F3: break;
        case F4: break;
        case F5: break;
        case F6: break;
        case F7: break;
        case F8: break;
        case F9: break;

        /* RETURN TO DOS */
        case F10: break;

        /* SPARE INPUT KEYS */

        case CTRL_F1: break;
        case CTRL_F2: break;
        case CTRL_F3: break;
        case CTRL_F4: break;
        case CTRL_F5: break;
        case CTRL_F6: break;
        case CTRL_F7: break;
        case CTRL_F8: break;
        case CTRL_F9: break;
        case CTRL_F10: break;

        case SHIFT_F1: break;
        case SHIFT_F2: break;
        case SHIFT_F3: break;
        case SHIFT_F4: break;
        case SHIFT_F5: break;
        case SHIFT_F6: break;
        case SHIFT_F7: break;
        case SHIFT_F8: break;
        case SHIFT_F9: break;
        case SHIFTF10: break;
```

```
            case ALT_F1: break;
            case ALT_F2: break;
            case ALT_F3: break;
            case ALT_F4: break;
            case ALT_F5: break;
            case ALT_F6: break;
            case ALT_F7: break;
            case ALT_F8: break;
            case ALT_F9: break;
            case ALT_F10: break;

            /* SCREEN MOVEMENT KEYS */

            case HOME: break;
            case UP_ARROW: break;
            case PG_UP: break;
            case LEFT_ARROW: break;
            case RIGHT_ARROW: break;
            case END: break;
            case DOWN_ARROW: break;
            case PG_DN: break;

            case CTRL_HOME: break;
            case CTRL_PG_UP: break;
            case CTRL_LEFT_ARROW: break;
            case CTRL_RIGHT_ARROW: break;
            case CTRL_END: break;
            case CTRL_PGDN: break;

            case INS: break;
            case DEL: break;

            default: break;
           }

        /* RETURN TO DOS */
        if ( key == F10 )
           break;

      }

    /* CLOSE THE GRAPHICS SYSTEM */
    closegraphics();

   } /* END MAIN */

/*************************************************************
                          END MAIN.C
 *************************************************************/
```

```
/*************************************************************
 *  HEADER FILE: UTILS.H                                     *
 *  WRITTEN BY: MARVIN A. WYANT, JR.                         *
 *  DATE OF LAST MODIFICATION: 11 MARCH 1988                 *
 *  PURPOSE: A LIBRARY OF STANDARD UTILITIES NOT INCLUDED    *
 *           IN THE LATTICE-C PACKAGE                        *
 *************************************************************/

#include "dos.h"  /* A STANDARD LATTICE-C HEADER FILE */

/*  CLEARS THE SCREEN WHEN NOT IN THE GRAPHICS MODE */

clrscr()
    {
     union REGS reg;

     reg.x.ax = ( 10 << 8 );
     reg.h.bh = BLACK;
     reg.x.cx = ( 0 << 8 );
     reg.x.dx = ( 24 << 8 ) + 79;
     int86( 0x10, &reg, &reg );

     gotoxy( 0, 0 );
    } /* END CLRSCR */

/* POSITIONS THE SYSTEM'S TEXT CURSOR AT AN X,Y COORDINATE
   ON THE SCREEN */

gotoxy( col, row )
    int col, row;
    {
     union REGS reg;

     reg.h.ah = 2;
     reg.h.bh = 0;
     reg.x.dx = ( row << 8 ) + col;
     int86( 0x10, &reg, &reg );
    } /* END GOTOXY */
```

```
/* GETS THE VALUES OF THE FUNCTION AND CURSOR KEYS */

int getkey()
    {
     int key1;
     int key2;
     do{
        key1 = getch();
        if ( key1 == 0 )
           {
            key2 = getch();
            return ( key2 );
           }
       } while ( key1 != 0 );
    } /* END GETKEY */

/*************************************************************
                          END UTILS.H
 *************************************************************/
```

```
/*********************************************************
 *  HEADER FILE: ICONS.H                                 *
 *  WRITTEN BY: MARVIN A. WYANT, JR.                     *
 *  DATE OF LAST MODIFICATION: 11 MARCH 1988             *
 *  PURPOSE: CONTAINS A LIBRARY OF FUNCTIONS USED TO DRAW *
 *           THE VARIOUS MODEL ELEMENT ICONS AND THE     *
 *           PLACEHOLDER ICON.  EACH FUNCTION REQUIRES   *
 *           THE X- AND Y-COORDINATES AND THE MODEL      *
 *           ELEMENT NAME.                               *
 *********************************************************/

/* DRAWS THE PRIMITIVE ENTITY ICON */

draw_pe( x, y, node_name )
     float x, y; /* BOTTOM-RIGHT COORDINATE OF THE ICON */
     char node_name[9]; /* NAME OF THE MODEL ELEMENT */
   {
    int color;
    int foreground;
    int background;
    float x1,x2,y1,y2;
    char label[3];
    x1 = x;
    x2 = x + 34.0;
    y1 = y + 16.0;
    y2 = y + 50.0;

    /* SET THE COLOR OF THE ICON */
    color = GREEN;
    setcolor( &color );

    /* DRAW THE ICON */
    bar( &x1, &y1, &x2, &y2 );

    /* SET ICON OUTLINE COLOR */
    color = BLACK;
    setcolor( &color );

    /* OUTLINE THE ICON */
    box( &x1, &y1, &x2, &y2 );

    x1 = x1 + 10.0;
    y1 = y1 + 13.0;

    /* SET THE TEXT COLOR FOR WRITING WITHIN THE ICON */
    foreground = BLACK;
    background = GREEN;
```

```
  /* LABEL THE ICON */
  settextclr( &foreground, &background );
  movtcurabs( &x1, &y1 );

  strcpy( label, "PE" );
  text( label );

  x1 = x1 + 30.0;
  y1 = y1 - 13.0;
  movtcurabs( &x1, &y1 );

  /* SET THE TEXT COLOR FOR WRITING THE MODEL ELEMENT
     NAME */
  background = WHITE;
  settextclr( &foreground, &background );

  /* WRITE THE NAME OF THE ICON */
  text( node_name );

  deltcur();
} /* END DRAW_PE */
```

```
/* DRAWS THE COMPLEX ENTITY ICON */

draw_ce( x, y, node_name )
     float x, y; /* BOTTOM-RIGHT COORDINATE OF THE ICON */
     char node_name[9]; /* NAME OF THE MODEL ELEMENT */
   {
    int color;
    int foreground;
    int background;
    int arrowsides = 3;
    int shapesides = 4;
    float x1,y1,dx,dy;

    /* RELATIVE X,Y COORDINATES FOR DRAWING THE ARROWHEAD */
    static float xarrow[] = { -4.0, 8.0, -4.0 };
    static float yarrow[] = { -8.0, 0.0, 8.0 };

    /* RELATIVE X,Y COORDINATES FOR DRAWING THE ICON */
    static float xshape[] = { -17.0, 17.0, 17.0, -17.0};
    static float yshape[] = { 17.0, 17.0, -17.0, -17.0};
    char label[3];
    x1 = x + 17.0;
    y1 = y + 16.0;
    movabs( &x1, &y1 );

    /* SET THE COLOR OF THE ICON */
    color = GREEN;

    /* DRAW THE ICON */
    polyfrel( xshape, yshape, &shapesides, &color );
    /* SET ICON OUTLINE COLOR */
    color = BLACK;
    setcolor( &color );

    /* OUTLINE THE ICON */
    polylnrel( xshape, yshape, &shapesides );

    /* DRAW THE ARROWHEAD LEADING INTO THE ICON */
    dx = 0.0;
    dy = -16.0;
    lnrel( &dx, &dy );
    movabs( &x1, &y1 );

    polyfrel( xarrow, yarrow, &arrowsides, &color );

    x1 = x1 - 7.0;
    y1 = y1 + 13.0;
```

```
  /* SET THE TEXT COLOR FOR WRITING WITHIN THE ICON */
  foreground = BLACK;
  background = GREEN;
  settextclr( &foreground, &background );
  movtcurabs( &x1, &y1 );

  /* LABEL THE ICON */
  strcpy( label, "CE" );
  text( label );

  x1 = x1 + 30.0;
  y1 = y1 - 13.0;
  movtcurabs( &x1, &y1 );

  /* SET THE TEXT COLOR FOR WRITING THE MODEL ELEMENT
     NAME */
  background = WHITE;
  settextclr( &foreground, &background );

  /* WRITE THE MODEL ELEMENT NAME */
  text( node_name );

  deltcur();
} /* END DRAW_CE */
```

```
/* DRAW THE ATTRIBUTE ICON */

draw_a( x, y, node_name )
     float x, y; /* BOTTOM-RIGHT COORDINATE OF THE ICON */
     char node_name[9]; /* NAME OF THE MODEL ELEMENT */
   {
    int color;
    int foreground;
    int background;
    int arrowsides = 3;
    float x1,y1,dx,dy;

    /* RELATIVE X,Y COORDINATES FOR DRAWING THE ARROWHEAD */
    static float xarrow[] = { -4.0, 8.0, -4.0 };
    static float yarrow[] = { -8.0, 0.0, 8.0 };

    /* RADIUS FOR DRAWING THE ICON */
    float radius = 19.0;
    char label[2];
    x1 = x + 17.0;
    y1 = y + 33.0;
    movabs( &x1, &y1 );

    /* SET THE COLOR OF THE ICON */
    color = BLUE;
    setcolor( &color );

    /* DRAW THE ICON */
    fcir( &radius );

    /* SET ICON OUTLINE COLOR */
    color = BLACK;
    setcolor( &color );

    /* OUTLINE THE ICON */
    cir( &radius );

    /* DRAW THE ARROWHEAD LEADING INTO THE ICON */
    y1 = y1 - 17.0;
    movabs( &x1, &y1 );
    dx = 0.0;
    dy = -16.0;
    lnrel( &dx, &dy );
    movabs( &x1, &y1 );

    polyfrel( xarrow, yarrow, &arrowsides, &color );

    x1 = x1 - 3.0;
    y1 = y1 + 13.0;
```

```
  /* SET THE TEXT COLOR FOR WRITING WITHIN THE ICON */
  foreground = WHITE;
  background = BLUE;
  settextclr( &foreground, &background );
  movtcurabs( &x1, &y1 );

  /* LABEL THE ICON */
  strcpy( label, "A" );
  text( label );

  x1 = x1 + 27.0;
  y1 = y1 - 13.0;
  movtcurabs( &x1, &y1 );

  /* SET THE TEXT COLOR FOR WRITING THE MODEL ELEMENT
     NAME */
  foreground = BLACK;
  background = WHITE;
  settextclr( &foreground, &background );

  /* WRITE THE MODEL ELEMENT NAME */
  text( node_name );

  deltcur();
} /* END DRAW_A */
```

```
/* DRAW THE VARIABLE ATTRIBUTE ICON */

draw_va( x, y, node_name )
     float x, y; /* BOTTOM-RIGHT COORDINATE OF THE ICON */
     char node_name[9]; /* NAME OF THE MODEL ELEMENT */
   {
    int color;
    int foreground;
    int background;
    int arrowsides = 3;
    int shapesides = 6;
    float x1,y1,dx,dy;

    /* RELATIVE COORDINATES FOR DRAWING THE ARROWHEAD */
    static float xarrow[] = { -4.0, 8.0, -4.0 };
    static float yarrow[] = { -8.0, 0.0, 8.0 };

    /* RELATIVE COORDINATES FOR DRAWING THE ICON */
    static float xshape[] =
                  { -6.0, -11.0, 17.0, 17.0, -11.0, -6.0 };
    static float yshape[] =
                  { 0.0, 20.0, 14.0, -14.0, -20.0, 0.0 };
    char label[3];
    x1 = x + 17.0;
    y1 = y + 16.0;
    movabs( &x1, &y1 );
    /* SET THE COLOR OF THE ICON */
    color = BLUE;
    /* DRAW THE ICON */
    polyfrel( xshape, yshape, &shapesides, &color );

    /* SET ICON OUTLINE COLOR */
    color = BLACK;
    setcolor( &color );

    /* DRAW THE OUTLINE */
    polylnrel( xshape, yshape, &shapesides );

    /* DRAW THE ARROWHEAD LEADING INTO THE ICON */
    dx = 0.0;
    dy = -16.0;
    lnrel( &dx, &dy );
    movabs( &x1, &y1 );

    polyfrel( xarrow, yarrow, &arrowsides, &color );

    x1 = x1 - 7.0;
    y1 = y1 + 8.0;
```

```
  /* SET THE TEXT COLOR FOR WRITING WITHIN THE ICON */
  foreground = WHITE;
  background = BLUE;
  settextclr( &foreground, &background );
  movtcurabs( &x1, &y1 );

  /* LABEL THE ICON */
  strcpy( label, "VA" );
  text( label );

  x1 = x1 + 30.0;
  y1 = y1 - 8.0;
  movtcurabs( &x1, &y1 );

  /* SET THE TEXT COLOR FOR WRITING THE MODEL ELEMENT
     NAME */
  foreground = BLACK;
  background = WHITE;
  settextclr( &foreground, &background );

  /* WRITE THE MODEL ELEMENT NAME */
  text( node_name );

  deltcur();
} /* END DRAW_VA */
```

```
/* DRAW THE FUNCTION ICON */

draw_f( x, y, node_name )
     float x, y; /* BOTTOM-RIGHT COORDINATE OF ICON */
     char node_name[9]; /* MODEL ELEMENT NAME */
   {
    int color;
    int foreground;
    int background;
    int arrowsides = 3;
    int shapesides = 4;
    float x1,y1,dx,dy;

    /* RELATIVE X,Y COORDINATES FOR DRAWING THE ARROWHEAD */
    static float xarrow[] = { -4.0, 8.0, -4.0 };
    static float yarrow[] = { -8.0, 0.0, 8.0 };

    /* RELATIVE X,Y COORDINATES FOR DRAWING THE ICON */
    static float xshape[] = { -17.0, 17.0, 17.0, -17.0 };
    static float yshape[] = { 0.0, 34.0, -34.0, 0.0 };
    char label[2];
    x1 = x + 17.0;
    y1 = y + 16.0;
    movabs( &x1, &y1 );

    /* SET THE COLOR OF THE ICON */
    color = YELLOW;

    /* DRAW THE ICON */
    polyfrel( xshape, yshape, &shapesides, &color );

    /* SET ICON OUTLINE COLOR */
    color = BLACK;
    setcolor( &color );

    /* OUTLINE THE ICON */
    polylnrel( xshape, yshape, &shapesides );

    /* DRAW THE ARROWHEAD LEADING INTO THE ICON */
    dx = 0.0;
    dy = -16.0;
    lnrel( &dx, &dy );
    movabs( &x1, &y1 );

    polyfrel( xarrow, yarrow, &arrowsides, &color );

    x1 = x1 - 3.0;
    y1 = y1 + 8.0;
```

```
  /* SET THE TEXT COLOR FOR WRITING WITHIN THE ICON */
  foreground = BLACK;
  background = YELLOW;
  settextclr( &foreground, &background );
  movtcurabs( &x1, &y1 );

  /* LABEL THE ICON */
  strcpy( label, "F" );
  text( label );

  x1 = x1 + 27.0;
  y1 = y1 - 8.0;
  movtcurabs( &x1, &y1 );

  /* SET THE TEXT COLOR FOR WRITING THE MODEL ELEMENT
     NAME */
  background = WHITE;
  settextclr( &foreground, &background );

  /* WRITE THE MODEL ELEMENT NAME */
  text( node_name );

  deltcur();
} /* END DRAW_F */
```

```
/* DRAW THE TEST ICON */

draw_t( x, y, node_name )
     float x, y; /* BOTTOM-RIGHT COORDINATE OF THE ICON */
     char node_name[9]; /* NAME OF THE MODEL ELEMENT */
   {
    int color;
    int foreground;
    int background;
    int arrowsides = 3;
    int shapesides = 7;
    float x1,y1,dx,dy;

    /* RELATIVE X,Y COORDINATES FOR DRAWING THE ARROWHEAD */
    static float xarrow[] = { -4.0, 8.0, -4.0 };
    static float yarrow[] = { -8.0, 0.0, 8.0 };

    /* RELATIVE X,Y COORDINATES FOR DRAWING THE ICON */
    static float xshape[] =
             { -7.0, -10.0, 10.0, 14.0, 10.0, -10.0, -7.0 };
    static float yshape[] =
             { 0.0, 17.0, 17.0, 0.0, -17.0, -17.0, 0.0 };
    char label[2];
    x1 = x + 17.0;
    y1 = y + 16.0;
    movabs( &x1, &y1 );

    /* SET THE COLOR OF THE ICON */
    color = RED;

    /* DRAW THE ICON */
    polyfrel( xshape, yshape, &shapesides, &color );

    /* SET ICON OUTLINE COLOR */
    color = BLACK;
    setcolor( &color );

    /* OUTLINE THE ICON */
    polylnrel( xshape, yshape, &shapesides );

    /* DRAW THE ARROWHEAD LEADING INTO THE ICON */
    dx = 0.0;
    dy = -16.0;
    lnrel( &dx, &dy );
    movabs( &x1, &y1 );

    polyfrel( xarrow, yarrow, &arrowsides, &color );

    x1 = x1 - 3.0;
    y1 = y1 + 13.0;
```

```
    /* SET THE TEXT COLOR FOR WRITING WITHIN THE ICON */
    foreground = WHITE;
    background = RED;
    settextclr( &foreground, &background );
    movtcurabs( &x1, &y1 );

    /* LABEL THE ICON */
    strcpy( label, "T" );
    text( label );

    x1 = x1 + 27.0;
    y1 = y1 - 13.0;
    movtcurabs( &x1, &y1 );

    /* SET THE TEXT COLOR FOR WRITING THE MODEL ELEMENT
       NAME */
    foreground = BLACK;
    background = WHITE;
    settextclr( &foreground, &background );

    /* WRITE THE MODEL ELEMENT NAME */
    text( node_name );

    deltcur();
  } /* END DRAW_T */

/* DRAW THE PLACEHOLDER ICON */

draw_space( x, y )
    float x, y; /* BOTTOM-RIGHT COORDINATE OF ICON */
    {
     float x1, y1;
     int color;

     /* SET COLOR FOR LINE THROUGH PLACEHOLDER ICON */
     color = BLACK;
     setcolor( &color );

     /* DRAW THE LINE */
     x1 = x + 17.0;
     y1 = y;
     movabs( &x1, &y1 );
     y1 = y1 + 50.0;
     lnabs( &x1, &y1 );
    } /* END DRAW_SPACE */

/*************************************************************
                        END ICONS.H
 *************************************************************/
```

```
/*************************************************************
 *  HEADER FILE: INITDISP.H                                  *
 *  WRITTEN BY: MARVIN A. WYANT, JR.                         *
 *  DATE OF LAST MODIFICATION: 11 MARCH 1988                 *
 *  PURPOSE: TO DETERMINE WHAT GRAPHICS DEVICE IS            *
 *           INSTALLED AND INITIALIZE THE SYSTEM SCREEN      *
 *************************************************************/

/* INITIALIZE THE SYSTEM SCREEN */

initdisplay()
   {
    int mode;
    char device_name[13];
    int color;
    int foreground;
    int background;
    int border;
    float x1;
    float y1;
    float x2;
    float y2;
    char message[25];

    /* SELECT THE APPROPRIATE GRAPHICS DEVICE */
    select_device( device_name, &mode );

    /* INSTALL THE GRAPHICS DIVICE DRIVER */
    setdev( device_name );

    /* INITIALIZE THE GRAPHICS SYSTEM */
    initgraphics( &mode );

    /* SET THE WORLD COORDINATE SYSTEM */
    x1 = 0.0; y1 = 0.0; x2 = 639.0; y2 = 384.0;
    setworld( &x1 , &y1 , &x2 , &y2 );

    /* INITIALIZE THE SYSTEM SCREEN */

    /* SET THE COLOR FOR THE INFORMATION AND QUICK
       REFERENCE LINES (THE SCREEN BORDER) */
    color = BROWN;
    setcolor( &color );

    /* SET THE TEXT COLOR FOR INFORMATION AND QUICK
       REFERENCE TEXT */
    foreground = WHITE;
    background = BROWN;
    settextclr( &foreground, &background );

    /* CLEAR THE SCREEN */
    clr();
```

```
    /* WRITE THE QUICK REFERENCE COMMANDS */
    x1 = 10.0;
    y1 = 3.0;
    movtcurabs( &x1, &y1 );
    strcpy( message, "F1 HELP   F10 QUIT" );
    text( message );

    /* VIEWPORT COORDINATES */
    x1=0.0;
    y1=0.04;
    x2=1.0;
    y2=0.96;

    /* SET COLOR FOR THE VIEWPORT */
    border = BROWN;
    background = WHITE;

    /* SET THE VIEWPORT */
    setviewport( &x1,&y1,&x2,&y2,&border,&background );
   } /* END INITDISPLAY */

/* GETS THE SYSTEM GRAPHICS DEVICE FROM THE USER */

select_device( device_name, mode)
    char device_name[13];
    int *mode;
    {
     /* IBM CGA CARD */
     static char device1[] = "HALOIBM.DEV";
     /* GENERIC CGA CARD */
     static char device2[] = "HALOIBMG.DEV";
     /* IBM EGA CARD */
     static char device3[] = "HALOIBME.DEV";
     /* SIGMA DESIGNS 400 EGA CARD */
     static char device4[] = "HALOSIGM.DEV";

     int device;

     do {
         /* WRITE THE POSSIBLE GRAPHICS DEVICE OPTIONS */
         clrscr();
         gotoxy( 18, 6 );
         printf(
           "Which graphics device do you have installed:" );
         gotoxy( 18, 8 );
         printf(
              "     1.  IBM Color Graphics Adapter (CGA)" );
         gotoxy( 18, 9 );
         printf( "     2.  IBM CGA Compatible" );
         gotoxy( 18, 10 );
         printf(
           "     3.  IBM Enhanced Graphics Adapter (EGA)" );
```

```
        gotoxy( 18, 11 );
        printf( "     4.  Sigma Designs Color 400" );
        gotoxy( 18, 14 );
        printf( "                Selection: " );

        /* GET THE USER'S INPUT */
        device = getche();
        } while ( device < 49 || device > 52 );

    clrscr();

    switch ( device )
       {
        /* GET THE NAME OF THE GRAPHICS DEVICE DRIVER */
        case 49:
             strcpy( device_name, device1 );
             *mode = 1;
             break;
        case 50:
             strcpy( device_name, device2 );
             *mode = 1;
             break;
        case 51:
             strcpy( device_name, device3 );
             *mode = 4;
             break;
        case 52:
             strcpy( device_name, device4 );
*mode = 3;
             break;
       }
   } /* END SELECT_DEVICE */

/*************************************************************
                      END INITDISP.H
 *************************************************************/
```

```
/*********************************************************
 *  HEADER FILE: HELP.H                                  *
 *  WRITTEN BY: MARVIN A. WYANT, JR.                     *
 *  DATE OF LAST MODIFICATION: 12 MARCH 1988             *
 *  PURPOSE: TO DISPLAY A HELP MENU                      *
 *********************************************************/

/* DISPLAY THE HELP MENU */

help()
    {
     /* AN ARRAY TO HOLD THE CONTENTS OF THE SCRREN
        COVERED BY THE HELP MENU */
     extern int window_array[30000];
     float x1, x2, x3, x4;
     float y1, y2, y3, y4;
     int border;
     int background;
     int mode;
     int g;

     /* AREA OF THE SCREEN TO BE SAVED */
     x1 = 0.0;
     y1 = 383.0;
     x2 = 639.0;
     y2 = 184.0;

     /* SAVE THE CURRENT SCREEN CONTENTS */
     movefrom( &x1, &y1, &x2, &y2, window_array );

     /* SET COLORS FOR MENU VIEWPORT */
     border = NOTHING;
     background = BLACK;

     /* MENU VIEWPORT COORDINATES */
     x3 = 0.0;
     y3 = 0.04;
     x4 = 1.0;
     y4 = 0.518;

     /* SET THE VIEWPORT FOR THE MENU */
     setviewport( &x3, &y3, &x4, &y4, &border,
                                          &background );

     /* DRAW A BOX FOR DISPLAYING THE TEXT WITHIN */
     border = WHITE;
     setcolor(&border);
     x3 = 4.0;
     y3 = 375.0;
     x4 = 634.0;
     y4 = 9.0;
     box(&x3,&y3,&x4,&y4);
```

```
    /* WRITE MENU TEXT */
    gotoxy(32,1);
    printf("HELP MENU");
    gotoxy(2,2);
    printf("F1 ..... HELP");
    gotoxy(2,3);
    printf("F9 ..... DEMO");
    gotoxy(2,4);
    printf("F10 .... RETURN TO DOS");
    gotoxy(30,12);
    printf("<ESC> to EXIT");

    /* EXIT FROM THE MENU */
    do  {
         g = getch();
         if ( g == 27 )
             {
              /* COORDINATES FOR SYSTEM VIEWPORT */
              x3 = 0.0;
              y3 = 0.04;
              x4 = 1.0;
              y4 = 0.96;

              border = NOTHING;
              background = NOTHING;

              /* RESTORE THE SYSTEM VIEWPORT */
              setviewport( &x3, &y3, &x4, &y4, &border,
                                              &background );

              /* RESTORE SCREEN COVERED BY MENU VIEWPORT */
              mode = 1;
              moveto( &x1, &y1, window_array, &mode );
             }
        } while (g != 27);
   } /* END HELP */

/*************************************************************
                          END HELP.H
 *************************************************************/
```

```
/*********************************************************
 *  HEADER FILE: DRAWGEN.H                               *
 *  WRITTEN BY: MARVIN A. WYANT, JR.                     *
 *  DATE OF LAST MODIFICATION: 12 MARCH 1988             *
 *  PURPOSE: TO DRAW A MODEL'S GENUS GRAPH               *
 *********************************************************/

/* DRAW THE GENUS GRAPH */

draw_genus(relptr, posptr, temptr)
    /* POINTER TO RELSHIP DATA STRUCTURES */
    RELSHP *relptr;
    /* POINTER TO ENTITY DATA STRUCTURES */
    POSIT *posptr;
    /* POINTER TO TEMPORARY ENTITY DATA STRUCTURES */
    POSIT *temptr;
    {
     int size1;
     int size2;
     int color;
     int i;
     int count1, count2, count3, count4, count5;
     int level_counter;
     int loop1, loop2;
     int test;
     int compare;
     int used_array[50];

     strcpy(( relptr + 1 ) -> e1name, "SUPPLY" );
     strcpy(( relptr + 1 ) -> e1type, "A" );
     strcpy(( relptr + 1 ) -> e2name, "PLANT" );
     strcpy(( relptr + 1 ) -> e2type, "PE" );
     strcpy(( relptr + 2 ) -> e1name, "LINK" );
     strcpy(( relptr + 2 ) -> e1type, "CE" );
     strcpy(( relptr + 2 ) -> e2name, "PLANT" );
     strcpy(( relptr + 2 ) -> e2type, "PE" );
     strcpy(( relptr + 3 ) -> e1name, "LINK" );
     strcpy(( relptr + 3 ) -> e1type, "CE" );
     strcpy(( relptr + 3 ) -> e2name, "CUSTOMER" );
     strcpy(( relptr + 3 ) -> e2type, "PE" );
     strcpy(( relptr + 4 ) -> e1name, "DEMAND" );
     strcpy(( relptr + 4 ) -> e1type, "A" );
     strcpy(( relptr + 4 ) -> e2name, "CUSTOMER" );
     strcpy(( relptr + 4 ) -> e2type, "PE" );
     strcpy(( relptr + 5 ) -> e1name, "T:SUP" );
     strcpy(( relptr + 5 ) -> e1type, "T" );
     strcpy(( relptr + 5 ) -> e2name, "SUPPLY" );
     strcpy(( relptr + 5 ) -> e2type, "A" );
     strcpy(( relptr + 6 ) -> e1name, "T:SUP" );
     strcpy(( relptr + 6 ) -> e1type, "T" );
     strcpy(( relptr + 6 ) -> e2name, "FLOW" );
     strcpy(( relptr + 6 ) -> e2type, "VA" );
```

```
 strcpy(( relptr + 7 ) -> e1name, "COST" );
 strcpy(( relptr + 7 ) -> e1type, "F" );
 strcpy(( relptr + 7 ) -> e2name, "LINK" );
 strcpy(( relptr + 7 ) -> e2type, "CE" );
 strcpy(( relptr + 8 ) -> e1name, "FLOW" );
 strcpy(( relptr + 8 ) -> e1type, "VA" );
 strcpy(( relptr + 8 ) -> e2name, "LINK" );
 strcpy(( relptr + 8 ) -> e2type, "CE" );
 strcpy(( relptr + 9 ) -> e1name, "TOT_COST" );
 strcpy(( relptr + 9 ) -> e1type, "F" );
 strcpy(( relptr + 9 ) -> e2name, "COST" );
 strcpy(( relptr + 9 ) -> e2type, "F" );
 strcpy(( relptr + 10 ) -> e1name, "TOT_COST" );
 strcpy(( relptr + 10 ) -> e1type, "F" );
 strcpy(( relptr + 10 ) -> e2name, "FLOW" );
 strcpy(( relptr + 10 ) -> e2type, "VA" );
 strcpy(( relptr + 11 ) -> e1name, "T:DEMAND" );
 strcpy(( relptr + 11 ) -> e1type, "T" );
 strcpy(( relptr + 11 ) -> e2name, "FLOW" );
 strcpy(( relptr + 11 ) -> e2type, "VA" );
 strcpy(( relptr + 12 ) -> e1name, "T:DEMAND" );
 strcpy(( relptr + 12 ) -> e1type, "T" );
 strcpy(( relptr + 12 ) -> e2name, "DEMAND" );
 strcpy(( relptr + 12 ) -> e2type, "A" );

size1 = 12;
count1 = 0;
count2 = 0;
level_counter = 1;

/* FIND ALL THE PRIMITIVE ENTITIES AND PLACE ON LEVEL
   ONE */
for ( loop1 = 1; loop1 <= size1; loop1 = loop1 + 1 )
    {
     compare = strcmp(( relptr + loop1 ) -> e2type,
                                                "PE" );
     if ( compare == 0 )
        {
         count1 = count1 + 1;
         strcpy(( posptr + count1 ) -> name,
                        ( relptr + loop1 ) -> e2name );
         strcpy(( posptr + count1 ) -> type,
                        ( relptr + loop1 ) -> e2type );
        ( posptr + count1 ) -> level = level_counter;
        }
     used_array[loop1] = 0;
    }
count4 = count1;
count3 = count1;
count5 = 1;
```

```
/* GET REMAINDER OF MODEL ELEMENTS AND PLACE ON
   APPROPRIATE LEVELS */
do {
    test = 1;
    level_counter = level_counter + 1;
    for ( loop1 = count5; loop1 <= count3;
                                    loop1 = loop1 + 1 )
        {
         for ( loop2 = 1; loop2 <= size1;
                                     loop2 = loop2 + 1)
             {
              compare = strcmp(
                          ( relptr + loop2 ) -> e2name,
                          ( posptr + loop1 ) -> name );
              if ( compare == 0 )
                 {
                  count2 = count2 + 1;
                  used_array[loop2] = 1;
                  count4 = count1 + count2;
                  strcpy(( posptr + count4 ) -> name,
                         ( relptr + loop2 ) -> e1name );
                  strcpy(( posptr + count4 ) -> type,
                         ( relptr + loop2 ) -> e1type );
                 ( posptr + count4 ) ->
                                  level = level_counter;
                 }
             }
        }

     /* TEST TO SEE IF ALL MODEL ELEMENTS HAVE BEEN
        ASSIGNED LEVELS */
     for ( i = 1; i <= size1; i = i + 1)
         {
          if ( used_array[i] == 0 )
             {
              test = 0;
             }
         }
     count3 = count4;
     count1 = count4;
     count5 = count4 - count2 +1;
     count2 = 0;
   } while ( test == 0 );

size2 = count3;
compress_array(posptr,temptr,&size2);

/* ASSIGN ELEMENT ICONS COORDINATES */
get_coordinates(posptr,&size2);
```

```
    /* CLEAR THE SYSTEM WORKSPACE */
    color = WHITE;
    setcolor( &color );
    clr();
    /* DRAW THE ICONS */
    draw_icons(posptr,size2);

    /* DRAW THE ARCS BETWEEN ICONS */
    draw_lines(relptr,size1,posptr,size2);
  } /* END DRAW_GENUS */

/* ELIMINATE DUPLICATE MODEL ELEMENTS WITHIN THE
   POSITIONING DATA STRUCTURE */

compress_array(posptr,temptr,size2)
    POSIT *posptr;
    POSIT *temptr;
    int *size2;
    {
     int count1;
     int count2;
     int loop1;
     int loop2;
     int compare1;
     int compare2;

     count1 = *size2;
     for (loop1 = 1; loop1 <= count1; loop1 = loop1 + 1)
         {
          for ( loop2 = loop1 + 1; loop2 <= count1;
                                        loop2 = loop2 + 1 )
              {
               compare1 = strcmp(( posptr + loop1 ) -> name,
                              ( posptr + loop2 ) -> name );
               compare2 = strcmp(( posptr + loop1 ) -> type,
                              ( posptr + loop2 ) -> type );
               if (( compare1 == 0 ) && ( compare2 == 0))
                  {
                   /* ASSIGN NECESSARY PLACEHOLDER ICONS */
                   if (( posptr + loop1 ) -> level ==
                              ( posptr + loop2 ) -> level )
                      {
                      ( posptr + loop2 ) -> level = 0;
                      }
                   else
                     {
                      strcpy((posptr + loop1) -> type,"SP");
                     }
                  }
              }
         }
     count2 = 0;
```

```
    /* ELIMINATE UNNECESSARY MODEL ELEMENT REFERENCES */
    for ( loop1 = 1; loop1 <= count1; loop1 = loop1 + 1 )
        {
         if (( posptr + loop1 ) -> level != 0 )
            {
             count2 = count2 + 1;
             strcpy(( temptr + count2 ) -> name,
                                ( posptr + loop1 ) -> name );
             strcpy(( temptr + count2 ) -> type,
                                ( posptr + loop1 ) -> type );
            ( temptr + count2 ) -> level =
                                 ( posptr + loop1 ) -> level;
            }
        }
    for ( loop1 = 1; loop1 <= count2; loop1 = loop1 + 1 )
        {
         strcpy(( posptr + loop1 ) -> name,
                                ( temptr + loop1 ) -> name );
         strcpy(( posptr + loop1 ) -> type,
                                ( temptr + loop1 ) -> type );
        ( posptr + loop1 ) -> level =
                                 ( temptr + loop1 ) -> level;
        }
     *size2 = count2;
    } /* END COMPRESS_ARRAY */

/* DRAW THE ICONS */

draw_icons(posptr,size)
    POSIT *posptr;
    int size;
    {
     int loop1;
     int comp_pe;
     int comp_ce;
     int comp_a;
     int comp_va;
     int comp_f;
     int comp_t;
     char name[9];
     char type[3];
     float xpos;
     float ypos;
```

```
/* DETERMINE THE ICON AND DRAW IT AT ITS ASSIGNED
   LOCATION */
for ( loop1 = 1; loop1 <= size; loop1 = loop1 + 1 )
    {
     strcpy( name, ( posptr + loop1 ) -> name );
     strcpy( type, ( posptr + loop1 ) -> type );
     xpos = ( posptr + loop1 ) -> xpos;
     ypos = ( posptr + loop1 ) -> ypos;
     comp_pe = strcmp( type,"PE" );
     comp_ce = strcmp( type,"CE" );
     comp_a = strcmp( type,"A" );
     comp_va = strcmp( type,"VA" );
     comp_f = strcmp( type,"F" );
     comp_t = strcmp( type,"T" );

     if ( comp_pe == 0 )
        {
         draw_pe( xpos, ypos, name );
        }
     else if ( comp_ce == 0 )
        {
         draw_ce( xpos, ypos, name );
        }
     else if ( comp_a == 0 )
        {
         draw_a( xpos, ypos, name );
        }
     else if ( comp_va == 0 )
        {
         draw_va( xpos, ypos, name );
        }

     else if ( comp_f == 0 )
        {
         draw_f( xpos, ypos, name );
        }
     else if ( comp_t == 0 )
        {
         draw_t( xpos, ypos, name );
        }
     else
        {
         draw_space( xpos, ypos );
        }
    }
} /* END DRAW_ICONS */
```

```
/* ASSIGN THE ICON COORDINATES */

get_coordinates( posptr, array_size )
    POSIT *posptr;
    int *array_size;
    {
     int loop1, loop2;
     int x;
     int size;
     int level_size[100];
     int max_level;
     int largest_level;
     float pic_width;
     float extra_space;
     float space_width;
     float xcursor, ycursor;

     size = *array_size;

     max_level =( posptr + size ) -> level;

     /* COUNT THE NUMBER OF ICONS IN EACH
        GENUS GRAPH LEVELS */
     for ( loop1 = 1; loop1 <= max_level; loop1 = loop1 + 1)
         {
          level_size[loop1] = 0;
         }

     for ( loop1 = 1; loop1 <= size; loop1 = loop1 + 1 )
         {
          x =( posptr + loop1 ) -> level;
          level_size[x] = level_size[x] + 1;
         }
     largest_level = 1;

     /* FIND THE MAXIMUM NUMBER OF ICONS ON A SINGLE
        LEVEL */
     for (loop1 =1;loop1 <= max_level - 1;loop1 = loop1 + 1)
         {
          x = loop1 + 1;
          if ( level_size[x] > level_size[loop1] )
             {
              largest_level = x;
             }
         }

     /* DETERMINE WIDTH OF GENUS GRAPH */
     if ( level_size[largest_level] < 6 )
        {
         pic_width = 6 * 102.0;
        }
```

```
      else
         {
          pic_width = level_size[largest_level] * 102.0;
         }
      ycursor = 32.5;

      /* ASSIGN EACH ICON ITS X,Y COORDINATES */
      for ( loop1 = 1; loop1 <= max_level; loop1 = loop1 + 1)
          {
           extra_space =
                   pic_width - ( level_size[loop1] * 102.0 );
           space_width =
                     extra_space / ( level_size[loop1] + 1 );
           xcursor = space_width;
           for ( loop2 = 1; loop2 <= size; loop2 = loop2 + 1)
               {
                if (( posptr + loop2 ) -> level == loop1 )
                   {
                   ( posptr + loop2 ) -> xpos = xcursor;
                   ( posptr + loop2 ) -> ypos = ycursor;
                    xcursor = xcursor + space_width + 102.0;
                   }
               }
           ycursor = ycursor + 90.0;
          }
     } /* END GET COORDINATES */

/* DRAW A LINE SEGEMENT FROM X1,Y1 TO X2,Y2 */

line_from( x1, y1, x2, y2 )
     float x1, y1, x2, y2;
     {
      int color;
      float xstart, ystart, xend, yend;

      xstart = x1 + 17.0;
      ystart = y1 + 50.0;
      xend  = x2 + 17.0;
      yend = y2;

      color = BLACK;
      setcolor( &color );

      movabs( &xstart, &ystart );
      lnabs( &xend, &yend );
     } /* END LINE_FROM */
```

```
/* DRAW RELATIONSHIP LINES */

draw_lines( relptr, sizegen, posptr, sizepos )
    RELSHP *relptr;
    int sizegen;
    POSIT *posptr;
    int sizepos;
    {
     int loop1, loop2, loop3;
     int compare;
     int base_level;
     int lev_diff;
     float x1, y1, x2, y2;
     char start_name[9];
     char end_name[9];

     /* DETERMINE ICONS FOR LINE TO BE DRAWN BETWEEN */
     for( loop1 = 1; loop1 <= sizegen; loop1 = loop1 + 1 )
          {
           strcpy(start_name, ( relptr + loop1 ) -> e2name);
           strcpy( end_name, ( relptr + loop1 ) -> e1name );
           for(loop2 = 1;loop2 <= sizepos;loop2 = loop2 + 1)
                {
                 compare = strcmp( start_name,
                                   ( posptr + loop2 ) -> name );
                 if( compare == 0 )
                     {
                      base_level =(posptr + loop2) -> level;
                      x1 = ( posptr + loop2 ) -> xpos;
                      y1 = ( posptr + loop2 ) -> ypos;
                      for(loop3 = loop2 + 1;loop3<= sizepos;
                                            loop3 = loop3 + 1 )
                           {
                            compare = strcmp( end_name,
                               ( posptr + loop3 ) -> name );
                            if ( compare == 0 )
                               {
          lev_diff = (posptr + loop3 ) -> level - base_level;

          /* DRAW LINES BETWEEN CONSECUTIVE LEVELS */
          if ( lev_diff == 1 )
             {
              x2 = ( posptr + loop3 ) -> xpos;
              y2 = ( posptr + loop3 ) -> ypos;
              line_from( x1, y1, x2, y2 );
             }
```

```
        /* DRAW LINES PASSING THROUGH LEVELS */
        else if ( lev_diff >= 2 )
            {
             x1 = x2;
             y1 = y2;
             x2 = ( posptr + loop3 ) -> xpos;
             y2 = ( posptr + loop3 ) -> ypos;
             line_from( x1, y1, x2, y2 );
            }
                                  }
                              }
                          }
                      }
          }
    } /* DRAW_LINES */

/*************************************************************
                        END DRAWGEN.H
 *************************************************************/
```

APPENDIX C

GUIDE TO USING THE GRAPHICAL USER INTERFACE PROGRAM

The following steps are required to use the graphical user interface program:

1. Copy the graphical user interface program (MMS.EXE) to the directory containing the ORACLE DBMS.

2. Load the ORACLE DBMS into extended memory.

   - When in the ORACLE directory, type 'ORACLE'.

3. Begin execution of the program by typing 'MMS'.

4. A menu for selecting the graphics device installed in your machine now appears on the monitor. Enter the number corresponding to the graphics device installed in your machine.

5. The system screen now appears on the screen. Several options are available at this point.

   a. Press F1 to display the help screen.

   b. Press F2 to draw the genus graph.

   c. Press F3 to display model element paragraphs (the genus graph must first be drawn).

      - Use the cursor keys to position the graphics cursor over the desired model element.

      - Press enter to display the paragraph information.

   d. Press F10 to exit the program.

LIST OF REFERENCES

1. Geoffrion, A. M., "An Introduction to Structured Modeling," Management Science, v. 33, May 1987.

2. Geoffrion, A. M., Structured Modeling, Research Monograph, University of California at Los Angeles Graduate School of Management Science, Los Angeles, California, 1985.

3. Dolk, D. R., Model Management and Structured Modeling: The Role of an Information Resource Directory, Paper No. 87-12, Naval Postgraduate School, Monterey, California, July, 1987.

4. Yourdon, E., Managing the Structured Techniques, Yourdon Press, 1986.

5. Luqi and Berzins, V., Rapid Prototyping of Real-Time Systems, Report No. NPS52-87-005, Naval Postgraduate School, Monterey, California, 1987.

6. Johnson, J. R., "A Prototypical Success Story," Datamation, v. 29, November 1983.

7. O'Dell, D., Design and Implementation of a Visual User Interface for a Structured Model Management System, M.S. Thesis, Naval Postgraduate School, Monterey, California, March 1988.

8. Atwick, B. A., Applied Concepts in Microcomputer Graphics, Prentice-Hall, Inc., 1984.

9. Monk, A., Fundamentals of Human-Computer Interaction, Academic Press, Inc., 1984.

10. Carey, T. T. and Mason, R. E. A., "Information System Prototyping: Techniques, Tools, and Methodologies," INFOR, v. 21, August, 1983.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information System Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Director, Information Systems (OP-945) Office of the Chief of Naval Operations Navy Department Washington, D. C. 20350-2000	1
4.	Superintendent, Naval Postgraduate School Computer Technology Programs, Code 37 Monterey, California 93943-5000	1
5.	Department of Computer Science and Operations Research Attn: Kendall E. Nygard 300 Minard Hall North Dakota State University Fargo, North Dakota 58105	1
6.	Professor Daniel R. Dolk, Code 54Dk Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943	1
7.	MAJ John B. Isett, Code 54Is Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943	1
8.	LT Marvin A. Wyant, Jr. c/o Mr. Karl Kurz 11833 Meadowspring Dallas, Texas 75218	15

Thesis
W937 Wyant
c.1 Design and implementa-
tion of a prototype gra-
phical user interface for
a model management system.